python-arango

May 20, 2021

Contents

Welcome to the documentation for **python-arango**, a Python driver for ArangoDB.

CHAPTER 1

Requirements

- ArangoDB version 3.7+
- Python version 3.6+

CHAPTER 2

Installation

```
~$ pip install python-arango
```

CHAPTER 3

Contents

3.1 Getting Started

Here is an example showing how **python-arango** client can be used:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient(hosts='http://localhost:8529')

# Connect to "_system" database as root user.
# This returns an API wrapper for "_system" database.
sys_db = client.db('_system', username='root', password='passwd')

# Create a new database named "test" if it does not exist.
if not sys_db.has_database('test'):
    sys_db.create_database('test')

# Connect to "test" database as root user.
# This returns an API wrapper for "test" database.
db = client.db('test', username='root', password='passwd')

# Create a new collection named "students" if it does not exist.
# This returns an API wrapper for "students" collection.
if db.has_collection('students'):
    students = db.collection('students')
else:
    students = db.create_collection('students')

# Add a hash index to the collection.
students.add_hash_index(fields=['name'], unique=False)

# Truncate the collection.
students.truncate()
```

(continues on next page)

(continued from previous page)

```
# Insert new documents into the collection.
students.insert({'name': 'jane', 'age': 19})
students.insert({'name': 'josh', 'age': 18})
students.insert({'name': 'jake', 'age': 21})

# Execute an AQL query. This returns a result cursor.
cursor = db.aql.execute('FOR doc IN students RETURN doc')

# Iterate through the cursor to retrieve the documents.
student_names = [document['name'] for document in cursor]
```

3.2 Databases

ArangoDB server can have an arbitrary number of **databases**. Each database has its own set of *collections* and *graphs*. There is a special database named `_system`, which cannot be dropped and provides operations for managing users, permissions and other databases. Most of the operations can only be executed by admin users. See *Users and Permissions* for more information.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
# This returns an API wrapper for "_system" database.
sys_db = client.db('_system', username='root', password='passwd')

# List all databases.
sys_db.databases()

# Create a new database named "test" if it does not exist.
# Only root user has access to it at time of its creation.
if not sys_db.has_database('test'):
    sys_db.create_database('test')

# Delete the database.
sys_db.delete_database('test')

# Create a new database named "test" along with a new set of users.
# Only "jane", "john", "jake" and root user have access to it.
if not sys_db.has_database('test'):
    sys_db.create_database(
        name='test',
        users=[
            {'username': 'jane', 'password': 'foo', 'active': True},
            {'username': 'john', 'password': 'bar', 'active': True},
            {'username': 'jake', 'password': 'baz', 'active': True},
        ],
    )

# Connect to the new "test" database as user "jane".
```

(continues on next page)

(continued from previous page)

```
db = client.db('test', username='jane', password='foo')

# Make sure that user "jane" has read and write permissions.
sys_db.update_permission(username='jane', permission='rw', database='test')

# Retrieve various database and server information.
db.name
db.username
db.version()
db.status()
db.details()
db.collections()
db.graphs()
db.engine()

# Delete the database. Note that the new users will remain.
sys_db.delete_database('test')
```

See *ArangoClient* and *StandardDatabase* for API specification.

3.3 Collections

A **collection** contains *documents*. It is uniquely identified by its name which must consist only of hyphen, underscore and alphanumeric characters. There are three types of collections in python-arango:

- **Standard Collection:** contains regular documents.
- **Vertex Collection:** contains vertex documents for graphs. See *here* for more details.
- **Edge Collection:** contains edge documents for graphs. See *here* for more details.

Here is an example showing how you can manage standard collections:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List all collections in the database.
db.collections()

# Create a new collection named "students" if it does not exist.
# This returns an API wrapper for "students" collection.
if db.has_collection('students'):
    students = db.collection('students')
else:
    students = db.create_collection('students')

# Retrieve collection properties.
students.name
students.db_name
students.properties()
students.revision()
```

(continues on next page)

(continued from previous page)

```
students.statistics()
students.checksum()
students.count()

# Perform various operations.
students.load()
students.unload()
students.truncate()
students.configure()

# Delete the collection.
db.delete_collection('students')
```

See *StandardDatabase* and *StandardCollection* for API specification.

3.4 Documents

In python-arango, a **document** is a Python dictionary with the following properties:

- Is JSON serializable.
- May be nested to an arbitrary depth.
- May contain lists.
- Contains the `_key` field, which identifies the document uniquely within a specific collection.
- Contains the `_id` field (also called the *handle*), which identifies the document uniquely across all collections within a database. This ID is a combination of the collection name and the document key using the format `{collection}/{key}` (see example below).
- Contains the `_rev` field. ArangoDB supports MVCC (Multiple Version Concurrency Control) and is capable of storing each document in multiple revisions. Latest revision of a document is indicated by this field. The field is populated by ArangoDB and is not required as input unless you want to validate a document against its current revision.

For more information on documents and associated terminologies, refer to ArangoDB manual. Here is an example of a valid document in "students" collection:

```
{
    '_id': 'students/bruce',
    '_key': 'bruce',
    '_rev': '_Wm3dzEi--_',
    'first_name': 'Bruce',
    'last_name': 'Wayne',
    'address': {
        'street' : '1007 Mountain Dr.',
        'city': 'Gotham',
        'state': 'NJ'
    },
    'is_rich': True,
    'friends': ['robin', 'gordon']
}
```

Edge documents (edges) are similar to standard documents but with two additional required fields `_from` and `_to`. Values of these fields must be the handles of "from" and "to" vertex documents linked by the edge document in

question (see *Graphs* for details). Edge documents are contained in *edge collections*. Here is an example of a valid edge document in "friends" edge collection:

```
{
    '_id': 'friends/001',
    '_key': '001',
    '_rev': '_Wm3d4le--_',
    '_from': 'students/john',
    '_to': 'students/jane',
    'closeness': 9.5
}
```

Standard documents are managed via collection API wrapper:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Create some test documents to play around with.
lola = {'_key': 'lola', 'GPA': 3.5, 'first': 'Lola', 'last': 'Martin'}
abby = {'_key': 'abby', 'GPA': 3.2, 'first': 'Abby', 'last': 'Page'}
john = {'_key': 'john', 'GPA': 3.6, 'first': 'John', 'last': 'Kim'}
emma = {'_key': 'emma', 'GPA': 4.0, 'first': 'Emma', 'last': 'Park'}

# Insert a new document. This returns the document metadata.
metadata = students.insert(lola)
assert metadata['_id'] == 'students/lola'
assert metadata['_key'] == 'lola'

# Check if documents exist in the collection in multiple ways.
assert students.has('lola') and 'john' not in students

# Retrieve the total document count in multiple ways.
assert students.count() == len(students) == 1

# Insert multiple documents in bulk.
students.import_bulk([abby, john, emma])

# Retrieve one or more matching documents.
for student in students.find({'first': 'John'}):
    assert student['_key'] == 'john'
    assert student['GPA'] == 3.6
    assert student['last'] == 'Kim'

# Retrieve a document by key.
students.get('john')

# Retrieve a document by ID.
students.get('students/john')

# Retrieve a document by body with "_id" field.
```

(continues on next page)

(continued from previous page)

```
students.get({'_id': 'students/john'})

# Retrieve a document by body with "_key" field.
students.get({'_key': 'john'})

# Retrieve multiple documents by ID, key or body.
students.get_many(['abby', 'students/lola', {'_key': 'john'}])

# Update a single document.
lola['GPA'] = 2.6
students.update(lola)

# Update one or more matching documents.
students.update_match({'last': 'Park'}, {'GPA': 3.0})

# Replace a single document.
emma['GPA'] = 3.1
students.replace(emma)

# Replace one or more matching documents.
becky = {'first': 'Becky', 'last': 'Solis', 'GPA': '3.3'}
students.replace_match({'first': 'Emma'}, becky)

# Delete a document by key.
students.delete('john')

# Delete a document by ID.
students.delete('students/lola')

# Delete a document by body with "_id" or "_key" field.
students.delete(emma)

# Delete multiple documents. Missing ones are ignored.
students.delete_many([abby, 'john', 'students/lola'])

# Iterate through all documents and update individually.
for student in students:
    student['GPA'] = 4.0
    student['happy'] = True
    students.update(student)
```

You can manage documents via database API wrappers also, but only simple operations (i.e. get, insert, update, replace, delete) are supported and you must provide document IDs instead of keys:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Create some test documents to play around with.
# The documents must have the "_id" field instead.
lola = {'_id': 'students/lola', 'GPA': 3.5}
abby = {'_id': 'students/abby', 'GPA': 3.2}
```

(continues on next page)

(continued from previous page)

```
john = {'_id': 'students/john', 'GPA': 3.6}
emma = {'_id': 'students/emma', 'GPA': 4.0}

# Insert a new document.
metadata = db.insert_document('students', lola)
assert metadata['_id'] == 'students/lola'
assert metadata['_key'] == 'lola'

# Check if a document exists.
assert db.has_document(lola) is True

# Get a document (by ID or body with "_id" field).
db.document('students/lola')
db.document(abby)

# Update a document.
lola['GPA'] = 3.6
db.update_document(lola)

# Replace a document.
lola['GPA'] = 3.4
db.replace_document(lola)

# Delete a document (by ID or body with "_id" field).
db.delete_document('students/lola')
```

See *StandardDatabase* and *StandardCollection* for API specification.

When managing documents, using collection API wrappers over database API wrappers is recommended as more operations are available and less sanity checking is performed under the hood.

3.5 Schema Validation

ArangoDB supports document validation using JSON schemas. You can use this feature by providing a schema during collection creation using the `schema` parameter.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

if db.has_collection('employees'):
    db.delete_collection('employees')

# Create a new collection named "employees" with custom schema.
my_schema = {
    'rule': {
        'type': 'object',
        'properties': {
            'name': {'type': 'string'},
```

(continues on next page)

(continued from previous page)

```
            'email': {'type': 'string'}
        },
        'required': ['name', 'email']
    },
    'level': 'moderate',
    'message': 'Schema Validation Failed.'
}
employees = db.create_collection(name='employees', schema=my_schema)

# Modify the schema.
employees.configure(schema=my_schema)

# Remove the schema.
employees.configure(schema={})
```

3.6 Indexes

Indexes can be added to collections to speed up document lookups. Every collection has a primary hash index on `_key` field by default. This index cannot be deleted or modified. Every edge collection has additional indexes on fields `_from` and `_to`. For more information on indexes, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Create a new collection named "cities".
cities = db.create_collection('cities')

# List the indexes in the collection.
cities.indexes()

# Add a new hash index on document fields "continent" and "country".
index = cities.add_hash_index(fields=['continent', 'country'], unique=True)

# Add new fulltext indexes on fields "continent" and "country".
index = cities.add_fulltext_index(fields=['continent'])
index = cities.add_fulltext_index(fields=['country'])

# Add a new skiplist index on field 'population'.
index = cities.add_skiplist_index(fields=['population'], sparse=False)

# Add a new geo-spatial index on field 'coordinates'.
index = cities.add_geo_index(fields=['coordinates'])

# Add a new persistent index on field 'currency'.
index = cities.add_persistent_index(fields=['currency'], sparse=True)

# Add a new TTL (time-to-live) index on field 'currency'.
```

(continues on next page)

(continued from previous page)

```
index = cities.add_ttl_index(fields=['ttl'], expiry_time=200)

# Indexes may be added with a name that can be referred to in AQL queries.
index = cities.add_hash_index(fields=['country'], name='my_hash_index')

# Delete the last index from the collection.
cities.delete_index(index['id'])
```

See *StandardCollection* for API specification.

3.7 Graphs

A **graph** consists of vertices and edges. Vertices are stored as documents in *vertex collections* and edges stored as documents in *edge collections*. The collections used in a graph and their relations are specified with *edge definitions*. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List existing graphs in the database.
db.graphs()

# Create a new graph named "school" if it does not already exist.
# This returns an API wrapper for "school" graph.
if db.has_graph('school'):
    school = db.graph('school')
else:
    school = db.create_graph('school')

# Retrieve various graph properties.
school.name
school.db_name
school.vertex_collections()
school.edge_definitions()

# Delete the graph.
db.delete_graph('school')
```

3.7.1 Edge Definitions

An **edge definition** specifies a directed relation in a graph. A graph can have arbitrary number of edge definitions. Each edge definition consists of the following components:

- **From Vertex Collections:** contain "from" vertices referencing "to" vertices.
- **To Vertex Collections:** contain "to" vertices referenced by "from" vertices.
- **Edge Collection:** contains edges that link "from" and "to" vertices.

Here is an example body of an edge definition:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

Here is an example showing how edge definitions are managed:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
if db.has_graph('school'):
    school = db.graph('school')
else:
    school = db.create_graph('school')

# Create an edge definition named "teach". This creates any missing
# collections and returns an API wrapper for "teach" edge collection.
if not school.has_edge_definition('teach'):
    teach = school.create_edge_definition(
        edge_collection='teach',
        from_vertex_collections=['teachers'],
        to_vertex_collections=['teachers']
    )

# List edge definitions.
school.edge_definitions()

# Replace the edge definition.
school.replace_edge_definition(
    edge_collection='teach',
    from_vertex_collections=['teachers'],
    to_vertex_collections=['lectures']
)

# Delete the edge definition (and its collections).
school.delete_edge_definition('teach', purge=True)
```

3.7.2 Vertex Collections

A **vertex collection** contains vertex documents, and shares its namespace with all other types of collections. Each graph can have an arbitrary number of vertex collections. Vertex collections that are not part of any edge definition are called **orphan collections**. You can manage vertex documents via standard collection API wrappers, but using vertex collection API wrappers provides additional safeguards:

- All modifications are executed in transactions.
- If a vertex is deleted, all connected edges are also automatically deleted.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Create a new vertex collection named "teachers" if it does not exist.
# This returns an API wrapper for "teachers" vertex collection.
if school.has_vertex_collection('teachers'):
    teachers = school.vertex_collection('teachers')
else:
    teachers = school.create_vertex_collection('teachers')

# List vertex collections in the graph.
school.vertex_collections()

# Vertex collections have similar interface as standard collections.
teachers.properties()
teachers.insert({'_key': 'jon', 'name': 'Jon'})
teachers.update({'_key': 'jon', 'age': 35})
teachers.replace({'_key': 'jon', 'name': 'Jon', 'age': 36})
teachers.get('jon')
teachers.has('jon')
teachers.delete('jon')
```

You can manage vertices via graph API wrappers also, but you must use document IDs instead of keys where applicable.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Create a new vertex collection named "lectures" if it does not exist.
# This returns an API wrapper for "lectures" vertex collection.
if school.has_vertex_collection('lectures'):
    school.vertex_collection('lectures')
else:
    school.create_vertex_collection('lectures')

# The "_id" field is required instead of "_key" field (except for insert).
school.insert_vertex('lectures', {'_key': 'CSC101'})
school.update_vertex({'_id': 'lectures/CSC101', 'difficulty': 'easy'})
school.replace_vertex({'_id': 'lectures/CSC101', 'difficulty': 'hard'})
school.has_vertex('lectures/CSC101')
school.vertex('lectures/CSC101')
school.delete_vertex('lectures/CSC101')
```

See *Graph* and *VertexCollection* for API specification.

3.7.3 Edge Collections

An **edge collection** contains *edge documents*, and shares its namespace with all other types of collections. You can manage edge documents via standard collection API wrappers, but using edge collection API wrappers provides additional safeguards:

- All modifications are executed in transactions.
- Edge documents are checked against the edge definitions on insert.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Get the API wrapper for edge collection "teach".
if school.has_edge_definition('teach'):
    teach = school.edge_collection('teach')
else:
    teach = school.create_edge_definition(
        edge_collection='teach',
        from_vertex_collections=['teachers'],
        to_vertex_collections=['lectures']
    )

# Edge collections have a similar interface as standard collections.
teach.insert({
    '_key': 'jon-CSC101',
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101'
})
teach.replace({
    '_key': 'jon-CSC101',
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101',
    'online': False
})
teach.update({
    '_key': 'jon-CSC101',
    'online': True
})
teach.has('jon-CSC101')
teach.get('jon-CSC101')
teach.delete('jon-CSC101')

# Create an edge between two vertices (essentially the same as insert).
teach.link('teachers/jon', 'lectures/CSC101', data={'online': False})
```

(continues on next page)

(continued from previous page)

```
# List edges going in/out of a vertex.
teach.edges('teachers/jon', direction='in')
teach.edges('teachers/jon', direction='out')
```

You can manage edges via graph API wrappers also, but you must use document IDs instead of keys where applicable.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# The "_id" field is required instead of "_key" field.
school.insert_edge(
    collection='teach',
    edge={
        '_id': 'teach/jon-CSC101',
        '_from': 'teachers/jon',
        '_to': 'lectures/CSC101'
    }
)
school.replace_edge({
    '_id': 'teach/jon-CSC101',
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101',
    'online': False,
})
school.update_edge({
    '_id': 'teach/jon-CSC101',
    'online': True
})
school.has_edge('teach/jon-CSC101')
school.edge('teach/jon-CSC101')
school.delete_edge('teach/jon-CSC101')
school.link('teach', 'teachers/jon', 'lectures/CSC101')
school.edges('teach', 'teachers/jon', direction='in')
```

See *Graph* and *EdgeCollection* for API specification.

3.7.4 Graph Traversals

Graph traversals are executed via the *arango.graph.Graph.traverse()* method. Each traversal can span across multiple vertex collections, and walk over edges and vertices using various algorithms.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
```

(continues on next page)

(continued from previous page)

```
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Get API wrappers for "from" and "to" vertex collections.
teachers = school.vertex_collection('teachers')
lectures = school.vertex_collection('lectures')

# Get the API wrapper for the edge collection.:
teach = school.edge_collection('teach')

# Insert vertices into the graph.
teachers.insert({'_key': 'jon', 'name': 'Professor jon'})
lectures.insert({'_key': 'CSC101', 'name': 'Introduction to CS'})
lectures.insert({'_key': 'MAT223', 'name': 'Linear Algebra'})
lectures.insert({'_key': 'STA201', 'name': 'Statistics'})

# Insert edges into the graph.
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/CSC101'})
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/STA201'})
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/MAT223'})

# Traverse the graph in outbound direction, breath-first.
school.traverse(
    start_vertex='teachers/jon',
    direction='outbound',
    strategy='bfs',
    edge_uniqueness='global',
    vertex_uniqueness='global',
)
```

See *arango.graph.Graph.traverse()* for API specification.

3.8 AQL

ArangoDB Query Language (AQL) is used to read and write data. It is similar to SQL for relational databases, but without the support for data definition operations such as creating or deleting *databases*, *collections* or *indexes*. For more information, refer to ArangoDB manual.

3.8.1 AQL Queries

AQL queries are invoked from AQL API wrapper. Executing queries returns *result cursors*.

Example:

```
from arango import ArangoClient, AQLQueryKillError

# Initialize the ArangoDB client.
client = ArangoClient()
```

(continues on next page)

(continued from previous page)

```
# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Insert some test documents into "students" collection.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21}
])

# Get the AQL API wrapper.
aql = db.aql

# Retrieve the execution plan without running the query.
aql.explain('FOR doc IN students RETURN doc')

# Validate the query without executing it.
aql.validate('FOR doc IN students RETURN doc')

# Execute the query
cursor = db.aql.execute(
  'FOR doc IN students FILTER doc.age < @value RETURN doc',
  bind_vars={'value': 19}
)
# Iterate through the result cursor
student_keys = [doc['_key'] for doc in cursor]

# List currently running queries.
aql.queries()

# List any slow queries.
aql.slow_queries()

# Clear slow AQL queries if any.
aql.clear_slow_queries()

# Retrieve AQL query tracking properties.
aql.tracking()

# Configure AQL query tracking properties.
aql.set_tracking(
    max_slow_queries=10,
    track_bind_vars=True,
    track_slow_queries=True
)

# Kill a running query (this should fail due to invalid ID).
try:
    aql.kill('some_query_id')
except AQLQueryKillError as err:
    assert err.http_code == 404
    assert err.error_code == 1591
```

See *AQL* for API specification.

3.8.2 AQL User Functions

AQL User Functions are custom functions you define in Javascript to extend AQL functionality. They are somewhat similar to SQL procedures.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the AQL API wrapper.
aql = db.aql

# Create a new AQL user function.
aql.create_function(
    # Grouping by name prefix is supported.
    name='functions::temperature::converter',
    code='function (celsius) { return celsius * 1.8 + 32; }'
)
# List AQL user functions.
aql.functions()

# Delete an existing AQL user function.
aql.delete_function('functions::temperature::converter')
```

See *AQL* for API specification.

3.8.3 AQL Query Cache

AQL Query Cache is used to minimize redundant calculation of the same query results. It is useful when read queries are issued frequently and write queries are not.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the AQL API wrapper.
aql = db.aql

# Retrieve AQL query cache properties.
aql.cache.properties()

# Configure AQL query cache properties
aql.cache.configure(mode='demand', max_results=10000)
```

(continues on next page)

(continued from previous page)

```
# Clear results in AQL query cache.
aql.cache.clear()
```

See *AQLQueryCache* for API specification.

3.9 Simple Queries

Caution: There is no option to add a TTL (Time to live) or batch size optimizations to the Simple Queries due to how Arango is handling simple collection HTTP requests. Your request may time out and you'll see a CursorNextError exception. The AQL queries provide full functionality.

Here is an example of using ArangoDB's **simply queries**:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Get the IDs of all documents in the collection.
students.ids()

# Get the keys of all documents in the collection.
students.keys()

# Get all documents in the collection with skip and limit.
students.all(skip=0, limit=100)

# Find documents that match the given filters.
students.find({'name': 'Mary'}, skip=0, limit=100)

# Get documents from the collection by IDs or keys.
students.get_many(['id1', 'id2', 'key1'])

# Get a random document from the collection.
students.random()

# Update all documents that match the given filters.
students.update_match({'name': 'Kim'}, {'age': 20})

# Replace all documents that match the given filters.
students.replace_match({'name': 'Ben'}, {'age': 20})

# Delete all documents that match the given filters.
students.delete_match({'name': 'John'})
```

Here are all simple query (and other utility) methods available:

- *arango.collection.Collection.all()*
- *arango.collection.Collection.export()*
- *arango.collection.Collection.find()*
- *arango.collection.Collection.find_near()*
- *arango.collection.Collection.find_in_range()*
- *arango.collection.Collection.find_in_radius()*
- *arango.collection.Collection.find_in_box()*
- *arango.collection.Collection.find_by_text()*
- *arango.collection.Collection.get_many()*
- *arango.collection.Collection.ids()*
- *arango.collection.Collection.keys()*
- *arango.collection.Collection.random()*
- *arango.collection.StandardCollection.update_match()*
- *arango.collection.StandardCollection.replace_match()*
- *arango.collection.StandardCollection.delete_match()*
- *arango.collection.StandardCollection.import_bulk()*

3.10 Cursors

Many operations provided by python-arango (e.g. executing *AQL* queries) return result **cursors** to batch the network communication between ArangoDB server and python-arango client. Each HTTP request from a cursor fetches the next batch of results (usually documents). Depending on the query, the total number of items in the result set may or may not be known in advance.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Set up some test data to query against.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21},
    {'_key': 'Suzy', 'age': 23},
    {'_key': 'Dave', 'age': 20}
])

# Execute an AQL query which returns a cursor object.
cursor = db.aql.execute(
    'FOR doc IN students FILTER doc.age > @val RETURN doc',
```

(continues on next page)

(continued from previous page)

```
    bind_vars={'val': 17},
    batch_size=2,
    count=True
)

# Get the cursor ID.
cursor.id

# Get the items in the current batch.
cursor.batch()

# Check if the current batch is empty.
cursor.empty()

# Get the total count of the result set.
cursor.count()

# Flag indicating if there are more to be fetched from server.
cursor.has_more()

# Flag indicating if the results are cached.
cursor.cached()

# Get the cursor statistics.
cursor.statistics()

# Get the performance profile.
cursor.profile()

# Get any warnings produced from the query.
cursor.warnings()

# Return the next item from the cursor. If current batch is depleted, the
# next batch if fetched from the server automatically.
cursor.next()

# Return the next item from the cursor. If current batch is depleted, an
# exception is thrown. You need to fetch the next batch manually.
cursor.pop()

# Fetch the next batch and add them to the cursor object.
cursor.fetch()

# Delete the cursor from the server.
cursor.close()
```

See *Cursor* for API specification.

If the fetched result batch is depleted while you are iterating over a cursor (or while calling the method *arango.cursor.Cursor.next()*), python-arango automatically sends an HTTP request to the server to fetch the next batch (just-in-time style). To control exactly when the fetches occur, you can use methods *arango.cursor.Cursor.fetch()* and *arango.cursor.Cursor.pop()* instead.

Example:

```
from arango import ArangoClient

```

(continues on next page)

(continued from previous page)

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Set up some test data to query against.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21}
])

# If you iterate over the cursor or call cursor.next(), batches are
# fetched automatically from the server just-in-time style.
cursor = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
result = [doc for doc in cursor]

# Alternatively, you can manually fetch and pop for finer control.
cursor = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
while cursor.has_more(): # Fetch until nothing is left on the server.
    cursor.fetch()
while not cursor.empty(): # Pop until nothing is left on the cursor.
    cursor.pop()
```

3.11 Async API Execution

In **asynchronous API executions**, python-arango sends API requests to ArangoDB in fire-and-forget style. The server processes the requests in the background, and the results can be retrieved once available via *AsyncJob* objects.

Example:

```
import time

from arango import (
    ArangoClient,
    AQLQueryExecuteError,
    AsyncJobCancelError,
    AsyncJobClearError
)

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Begin async execution. This returns an instance of AsyncDatabase, a
# database-level API wrapper tailored specifically for async execution.
async_db = db.begin_async_execution(return_result=True)

# Child wrappers are also tailored for async execution.
async_aql = async_db.aql
async_col = async_db.collection('students')
```

(continues on next page)

(continued from previous page)

```
# API execution context is always set to "async".
assert async_db.context == 'async'
assert async_aql.context == 'async'
assert async_col.context == 'async'

# On API execution, AsyncJob objects are returned instead of results.
job1 = async_col.insert({'_key': 'Neal'})
job2 = async_col.insert({'_key': 'Lily'})
job3 = async_aql.execute('RETURN 100000')
job4 = async_aql.execute('INVALID QUERY')  # Fails due to syntax error.

# Retrieve the status of each async job.
for job in [job1, job2, job3, job4]:
    # Job status can be "pending", "done" or "cancelled".
    assert job.status() in {'pending', 'done', 'cancelled'}

    # Let's wait until the jobs are finished.
    while job.status() != 'done':
        time.sleep(0.1)

# Retrieve the results of successful jobs.
metadata = job1.result()
assert metadata['_id'] == 'students/Neal'

metadata = job2.result()
assert metadata['_id'] == 'students/Lily'

cursor = job3.result()
assert cursor.next() == 100000

# If a job fails, the exception is propagated up during result retrieval.
try:
    result = job4.result()
except AQLQueryExecuteError as err:
    assert err.http_code == 400
    assert err.error_code == 1501
    assert 'syntax error' in err.message

# Cancel a job. Only pending jobs still in queue may be cancelled.
# Since job3 is done, there is nothing to cancel and an exception is raised.
try:
    job3.cancel()
except AsyncJobCancelError as err:
    assert err.message.endswith(f'job {job3.id} not found')

# Clear the result of a job from ArangoDB server to free up resources.
# Result of job4 was removed from the server automatically upon retrieval,
# so attempt to clear it raises an exception.
try:
    job4.clear()
except AsyncJobClearError as err:
    assert err.message.endswith(f'job {job4.id} not found')

# List the IDs of the first 100 async jobs completed.
db.async_jobs(status='done', count=100)
```

(continues on next page)

(continued from previous page)

```
# List the IDs of the first 100 async jobs still pending.
db.async_jobs(status='pending', count=100)

# Clear all async jobs still sitting on the server.
db.clear_async_jobs()
```

Note: Be mindful of server-side memory capacity when issuing a large number of async requests in small time interval.

See *AsyncDatabase* and *AsyncJob* for API specification.

3.12 Batch API Execution

In **batch API executions**, requests to ArangoDB server are stored in client-side in-memory queue, and committed together in a single HTTP call. After the commit, results can be retrieved later from *BatchJob* objects.

Example:

```
from arango import ArangoClient, AQLQueryExecuteError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Begin batch execution via context manager. This returns an instance of
# BatchDatabase, a database-level API wrapper tailored specifically for
# batch execution. The batch is automatically committed when exiting the
# context. The BatchDatabase wrapper cannot be reused after commit.
with db.begin_batch_execution(return_result=True) as batch_db:

    # Child wrappers are also tailored for batch execution.
    batch_aql = batch_db.aql
    batch_col = batch_db.collection('students')

    # API execution context is always set to "batch".
    assert batch_db.context == 'batch'
    assert batch_aql.context == 'batch'
    assert batch_col.context == 'batch'

    # BatchJob objects are returned instead of results.
    job1 = batch_col.insert({'_key': 'Kris'})
    job2 = batch_col.insert({'_key': 'Rita'})
    job3 = batch_aql.execute('RETURN 100000')
    job4 = batch_aql.execute('INVALID QUERY')  # Fails due to syntax error.

# Upon exiting context, batch is automatically committed.
assert 'Kris' in students
assert 'Rita' in students
```

(continues on next page)

(continued from previous page)

```
# Retrieve the status of each batch job.
for job in batch_db.queued_jobs():
    # Status is set to either "pending" (transaction is not committed yet
    # and result is not available) or "done" (transaction is committed and
    # result is available).
    assert job.status() in {'pending', 'done'}

# Retrieve the results of successful jobs.
metadata = job1.result()
assert metadata['_id'] == 'students/Kris'

metadata = job2.result()
assert metadata['_id'] == 'students/Rita'

cursor = job3.result()
assert cursor.next() == 100000

# If a job fails, the exception is propagated up during result retrieval.
try:
    result = job4.result()
except AQLQueryExecuteError as err:
    assert err.http_code == 400
    assert err.error_code == 1501
    assert 'syntax error' in err.message

# Batch execution can be initiated without using a context manager.
# If return_result parameter is set to False, no jobs are returned.
batch_db = db.begin_batch_execution(return_result=False)
batch_db.collection('students').insert({'_key': 'Jake'})
batch_db.collection('students').insert({'_key': 'Jill'})

# The commit must be called explicitly.
batch_db.commit()
assert 'Jake' in students
assert 'Jill' in students
```

Note:

- Be mindful of client-side memory capacity when issuing a large number of requests in single batch execution.
- *BatchDatabase* and *BatchJob* instances are stateful objects, and should not be shared across multiple threads.
- *BatchDatabase* instance cannot be reused after commit.

See *BatchDatabase* and *BatchJob* for API specification.

3.13 Transactions

In **transactions**, requests to ArangoDB server are committed as a single, logical unit of work (ACID compliant).

Warning: New transaction REST API was added to ArangoDB version 3.5. In order to use it python-arango's own transaction API had to be overhauled in version 5.0.0. **The changes are not backward-compatible**: context

managers are no longer offered (you must always commit the transaction youself), method signatures are different when beginning the transaction, and results are returned immediately instead of job objects.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')
col = db.collection('students')

# Begin a transaction. Read and write collections must be declared ahead of
# time. This returns an instance of TransactionDatabase, database-level
# API wrapper tailored specifically for executing transactions.
txn_db = db.begin_transaction(read=col.name, write=col.name)

# The API wrapper is specific to a single transaction with a unique ID.
txn_db.transaction_id

# Child wrappers are also tailored only for the specific transaction.
txn_aql = txn_db.aql
txn_col = txn_db.collection('students')

# API execution context is always set to "transaction".
assert txn_db.context == 'transaction'
assert txn_aql.context == 'transaction'
assert txn_col.context == 'transaction'

# From python-arango version 5+, results are returned immediately instead
# of job objects on API execution.
assert '_rev' in txn_col.insert({'_key': 'Abby'})
assert '_rev' in txn_col.insert({'_key': 'John'})
assert '_rev' in txn_col.insert({'_key': 'Mary'})

# Check the transaction status.
txn_db.transaction_status()

# Commit the transaction.
txn_db.commit_transaction()
assert 'Abby' in col
assert 'John' in col
assert 'Mary' in col
assert len(col) == 3

# Begin another transaction. Note that the wrappers above are specific to
# the last transaction and cannot be reused. New ones must be created.
txn_db = db.begin_transaction(read=col.name, write=col.name)
txn_col = txn_db.collection('students')
assert '_rev' in txn_col.insert({'_key': 'Kate'})
assert '_rev' in txn_col.insert({'_key': 'Mike'})
assert '_rev' in txn_col.insert({'_key': 'Lily'})
assert len(txn_col) == 6
```

(continues on next page)

(continued from previous page)

```
# Abort the transaction
txn_db.abort_transaction()
assert 'Kate' not in col
assert 'Mike' not in col
assert 'Lily' not in col
assert len(col) == 3  # transaction is aborted so txn_col cannot be used
```

See *TransactionDatabase* for API specification.

Alternatively, you can use *arango.database.StandardDatabase.execute_transaction()* to run raw Javascript code in a transaction.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Execute transaction in raw Javascript.
result = db.execute_transaction(
    command='''
    function () {{
        var db = require('internal').db;
        db.students.save(params.student1);
        if (db.students.count() > 1) {
            db.students.save(params.student2);
        } else {
            db.students.save(params.student3);
        }
        return true;
    }}
    ''',
    params={
        'student1': {'_key': 'Lucy'},
        'student2': {'_key': 'Greg'},
        'student3': {'_key': 'Dona'}
    },
    read='students',  # Specify the collections read.
    write='students'  # Specify the collections written.
)
assert result is True
assert 'Lucy' in students
assert 'Greg' in students
assert 'Dona' not in students
```

3.14 Server Administration

Python-arango provides operations for server administration and monitoring. Most of these operations can only be performed by admin users via `_system` database.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db('_system', username='root', password='passwd')

# Retrieve the server version.
sys_db.version()

# Retrieve the server details.
sys_db.details()

# Retrieve the target DB version.
sys_db.required_db_version()

# Retrieve the database engine.
sys_db.engine()

# Retrieve the server time.
sys_db.time()

# Retrieve the server role in a cluster.
sys_db.role()

# Retrieve the server statistics.
sys_db.statistics()

# Read the server log.
sys_db.read_log(level="debug")

# Retrieve the log levels.
sys_db.log_levels()

# Set the log .
sys_db.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)

# Echo the last request.
sys_db.echo()

# Reload the routing collection.
sys_db.reload_routing()

# Retrieve server metrics.
sys_db.metrics()
```

Features available in enterprise edition only:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user using JWT authentication.
sys_db = client.db(
    '_system',
    username='root',
    password='passwd',
    auth_method='jwt'
)

# Retrieve JWT secrets.
sys_db.jwt_secrets()

# Hot-reload JWT secrets.
sys_db.reload_jwt_secrets()

# Rotate the user-supplied keys for encryption.
sys_db.encryption()
```

See *StandardDatabase* for API specification.

3.15 Users and Permissions

Python-arango provides operations for managing users and permissions. Most of these operations can only be performed by admin users via `_system` database.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db('_system', username='root', password='passwd')

# List all users.
sys_db.users()

# Create a new user.
sys_db.create_user(
    username='johndoe@gmail.com',
    password='first_password',
    active=True,
    extra={'team': 'backend', 'title': 'engineer'}
)

# Check if a user exists.
sys_db.has_user('johndoe@gmail.com')

# Retrieve details of a user.
```

(continues on next page)

(continued from previous page)

```
sys_db.user('johndoe@gmail.com')

# Update an existing user.
sys_db.update_user(
    username='johndoe@gmail.com',
    password='second_password',
    active=True,
    extra={'team': 'frontend', 'title': 'engineer'}
)

# Replace an existing user.
sys_db.replace_user(
    username='johndoe@gmail.com',
    password='third_password',
    active=True,
    extra={'team': 'frontend', 'title': 'architect'}
)

# Retrieve user permissions for all databases and collections.
sys_db.permissions('johndoe@gmail.com')

# Retrieve user permission for "test" database.
sys_db.permission(
    username='johndoe@gmail.com',
    database='test'
)

# Retrieve user permission for "students" collection in "test" database.
sys_db.permission(
    username='johndoe@gmail.com',
    database='test',
    collection='students'
)

# Update user permission for "test" database.
sys_db.update_permission(
    username='johndoe@gmail.com',
    permission='rw',
    database='test'
)

# Update user permission for "students" collection in "test" database.
sys_db.update_permission(
    username='johndoe@gmail.com',
    permission='ro',
    database='test',
    collection='students'
)

# Reset user permission for "test" database.
sys_db.reset_permission(
    username='johndoe@gmail.com',
    database='test'
)

# Reset user permission for "students" collection in "test" database.
sys_db.reset_permission(
```

(continues on next page)

(continued from previous page)

```
    username='johndoe@gmail.com',
    database='test',
    collection='students'
)
```

See *StandardDatabase* for API specification.

3.16 Tasks

ArangoDB can schedule user-defined Javascript snippets as one-time or periodic (re-scheduled after each execution) tasks. Tasks are executed in the context of the database they are defined in.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List all active tasks
db.tasks()

# Create a new task which simply prints parameters.
db.create_task(
    name='test_task',
    command='''
        var task = function(params){
            var db = require('@arangodb');
            db.print(params);
        }
        task(params);
    ''',
    params={'foo': 'bar'},
    offset=300,
    period=10,
    task_id='001'
)

# Retrieve details on a task by ID.
db.task('001')

# Delete an existing task by ID.
db.delete_task('001', ignore_missing=True)
```

Note: When deleting a database, any tasks that were initialized under its context remain active. It is therefore advisable to delete any running tasks before deleting the database.

Refer to *StandardDatabase* class for API specification.

3.17 Write-Ahead Log (WAL)

Write-Ahead Log (WAL) is a set of append-only files recording all writes on ArangoDB server. It is typically used to perform data recovery after a crash or synchronize slave databases with master databases in replicated environments. WAL operations can only be performed by admin users via `_system` database.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db('_system', username='root', password='passwd')

# Get the WAL API wrapper.
wal = sys_db.wal

# Configure WAL properties.
wal.configure(
    historic_logs=15,
    oversized_ops=False,
    log_size=30000000,
    reserve_logs=5,
    throttle_limit=0,
    throttle_wait=16000
)

# Retrieve WAL properties.
wal.properties()

# List WAL transactions.
wal.transactions()

# Flush WAL with garbage collection.
wal.flush(garbage_collect=True)

# Get the available ranges of tick values.
wal.tick_ranges()

# Get the last available tick value.
wal.last_tick()

# Get recent WAL operations.
wal.tail()
```

See `WriteAheadLog` for API specification.

3.18 Pregel

Python-arango provides support for **Pregel**, ArangoDB module for distributed iterative graph processing. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the Pregel API wrapper.
pregel = db.pregel

# Start a new Pregel job in "school" graph.
job_id = db.pregel.create_job(
    graph='school',
    algorithm='pagerank',
    store=False,
    max_gss=100,
    thread_count=1,
    async_mode=False,
    result_field='result',
    algorithm_params={'threshold': 0.000001}
)

# Retrieve details of a Pregel job by ID.
job = pregel.job(job_id)

# Delete a Pregel job by ID.
pregel.delete_job(job_id)
```

See *Pregel* for API specification.

3.19 Foxx

Python-arango provides support for **Foxx**, a microservice framework which lets you define custom HTTP endpoints to extend ArangoDB's REST API. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
db = client.db('_system', username='root', password='passwd')

# Get the Foxx API wrapper.
foxx = db.foxx

# Define the test mount point.
service_mount = '/test_mount'

# List services.
foxx.services()
```

(continues on next page)

(continued from previous page)

```
# Create a service using source on server.
foxx.create_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    development=True,
    setup=True,
    legacy=True
)

# Update (upgrade) a service.
service = db.foxx.update_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    teardown=True,
    setup=True,
    legacy=False
)

# Replace (overwrite) a service.
service = db.foxx.replace_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    teardown=True,
    setup=True,
    legacy=True,
    force=False
)

# Get service details.
foxx.service(service_mount)

# Manage service configuration.
foxx.config(service_mount)
foxx.update_config(service_mount, config={})
foxx.replace_config(service_mount, config={})

# Manage service dependencies.
foxx.dependencies(service_mount)
foxx.update_dependencies(service_mount, dependencies={})
foxx.replace_dependencies(service_mount, dependencies={})

# Toggle development mode for a service.
foxx.enable_development(service_mount)
foxx.disable_development(service_mount)

# Other miscellaneous functions.
foxx.readme(service_mount)
foxx.swagger(service_mount)
foxx.download(service_mount)
foxx.commit(service_mount)
foxx.scripts(service_mount)
```

(continues on next page)

(continued from previous page)

```
foxx.run_script(service_mount, 'setup', [])
foxx.run_tests(service_mount, reporter='xunit', output_format='xml')

# Delete a service.
foxx.delete_service(service_mount)
```

You can also manage Foxx services by using zip or Javascript files directly:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
db = client.db('_system', username='root', password='passwd')

# Get the Foxx API wrapper.
foxx = db.foxx

# Define the test mount point.
service_mount = '/test_mount'

# Create a service by providing a file directly.
foxx.create_service_with_file(
    mount=service_mount,
    filename='/home/user/service.zip',
    development=True,
    setup=True,
    legacy=True
)

# Update (upgrade) a service by providing a file directly.
foxx.update_service_with_file(
    mount=service_mount,
    filename='/home/user/service.zip',
    teardown=False,
    setup=True,
    legacy=True,
    force=False
)

# Replace a service by providing a file directly.
foxx.replace_service_with_file(
    mount=service_mount,
    filename='/home/user/service.zip',
    teardown=False,
    setup=True,
    legacy=True,
    force=False
)

# Delete a service.
foxx.delete_service(service_mount)
```

See *Foxx* for API specification.

3.20 Views and ArangoSearch

Python-arango supports **view** management. For more information on view properties, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Retrieve list of views.
db.views()

# Create a view.
db.create_view(
    name='foo',
    view_type='arangosearch',
    properties={
        'cleanupIntervalStep': 0,
        'consolidationIntervalMsec': 0
    }
)

# Rename a view.
db.rename_view('foo', 'bar')

# Retrieve view properties.
db.view('bar')

# Partially update view properties.
db.update_view(
    name='bar',
    properties={
        'cleanupIntervalStep': 1000,
        'consolidationIntervalMsec': 200
    }
)

# Replace view properties. Unspecified ones are reset to default.
db.replace_view(
    name='bar',
    properties={'cleanupIntervalStep': 2000}
)

# Delete a view.
db.delete_view('bar')
```

Python-arango also supports **ArangoSearch** views.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
```

(continues on next page)

(continued from previous page)

```
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Create an ArangoSearch view.
db.create_arangosearch_view(
    name='arangosearch_view',
    properties={'cleanupIntervalStep': 0}
)

# Partially update an ArangoSearch view.
db.update_arangosearch_view(
    name='arangosearch_view',
    properties={'cleanupIntervalStep': 1000}
)

# Replace an ArangoSearch view.
db.replace_arangosearch_view(
    name='arangosearch_view',
    properties={'cleanupIntervalStep': 2000}
)

# ArangoSearch views can be retrieved or deleted using regular view API
db.view('arangosearch_view')
db.delete_view('arangosearch_view')
```

For more information on the content of view **properties**, see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Refer to *StandardDatabase* class for API specification.

3.21 Analyzers

Python-arango supports **analyzers**. For more information on analyzers, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Retrieve list of analyzers.
db.analyzers()

# Create an analyzer.
db.create_analyzer(
    name='test_analyzer',
    analyzer_type='identity',
    properties={},
    features=[]
```

(continues on next page)

(continued from previous page)

```
)

# Delete an analyzer.
db.delete_analyzer('test_analyzer', ignore_missing=True)
```

Refer to *StandardDatabase* class for API specification.

3.22 Multithreading

There are a few things you should consider before using python-arango in a multithreaded (or multiprocess) architecture.

3.22.1 Stateful Objects

Instances of the following classes are considered *stateful*, and should not be accessed across multiple threads without locks in place:

- *BatchDatabase* (see *Batch API Execution*)
- *BatchJob* (see *Batch API Execution*)
- *Cursor* (see *Cursors*)

3.22.2 HTTP Sessions

When *ArangoClient* is initialized, a requests.Session instance is created per ArangoDB host connected. HTTP requests to a host are sent using only its corresponding session. For more information on how to override this behaviour, see *HTTP Clients*.

Note that a requests.Session object may not always be thread-safe. Always research your use case!

3.23 Error Handling

All python-arango exceptions inherit *arango.exceptions.ArangoError*, which splits into subclasses *arango.exceptions.ArangoServerError* and *arango.exceptions.ArangoClientError*.

3.23.1 Server Errors

arango.exceptions.ArangoServerError exceptions lightly wrap non-2xx HTTP responses coming from ArangoDB. Each exception object contains the error message, error code and HTTP request response details.

Example:

```
from arango import ArangoClient, ArangoServerError, DocumentInsertError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')
```

(continues on next page)

(continued from previous page)

```
# Get the API wrapper for "students" collection.
students = db.collection('students')

try:
    students.insert({'_key': 'John'})
    students.insert({'_key': 'John'})  # duplicate key error

except DocumentInsertError as exc:

    assert isinstance(exc, ArangoServerError)
    assert exc.source == 'server'

    exc.message           # Exception message usually from ArangoDB
    exc.error_message     # Raw error message from ArangoDB
    exc.error_code        # Error code from ArangoDB
    exc.url               # URL (API endpoint)
    exc.http_method       # HTTP method (e.g. "POST")
    exc.http_headers      # Response headers
    exc.http_code         # Status code (e.g. 200)

    # You can inspect the ArangoDB response directly.
    response = exc.response
    response.method       # HTTP method (e.g. "POST")
    response.headers      # Response headers
    response.url          # Full request URL
    response.is_success   # Set to True if HTTP code is 2XX
    response.body         # JSON-deserialized response body
    response.raw_body     # Raw string response body
    response.status_text  # Status text (e.g "OK")
    response.status_code  # Status code (e.g. 200)
    response.error_code   # Error code from ArangoDB

    # You can also inspect the request sent to ArangoDB.
    request = exc.request
    request.method        # HTTP method (e.g. "post")
    request.endpoint      # API endpoint starting with "/_api"
    request.headers       # Request headers
    request.params        # URL parameters
    request.data          # Request payload
```

See *Response* and *Request* for reference.

3.23.2 Client Errors

arango.exceptions.ArangoClientError exceptions originate from python-arango client itself. They do not contain error codes nor HTTP request response details.

Example:

```
from arango import ArangoClient, ArangoClientError, DocumentParseError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
```

(continues on next page)

(continued from previous page)

```
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

try:
    students.get({'_id': 'invalid_id'})  # malformed document

except DocumentParseError as exc:

    assert isinstance(exc, ArangoClientError)
    assert exc.source == 'client'

    # Only the error message is set.
    error_message = exc.message
    assert exc.error_code is None
    assert exc.error_message is None
    assert exc.url is None
    assert exc.http_method is None
    assert exc.http_code is None
    assert exc.http_headers is None
    assert exc.response is None
    assert exc.request is None
```

3.23.3 Exceptions

Below are all exceptions from python-arango.

exception arango.exceptions.**ArangoError**
Base class for all exceptions in python-arango.

exception arango.exceptions.**ArangoClientError**(*msg: str*)
Base class for errors originating from python-arango client.

Parameters **msg** (*str*) – Error message.

Variables

- **source** (*str*) – Source of the error (always set to "client").
- **message** (*str*) – Error message.

exception arango.exceptions.**ArangoServerError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Base class for errors originating from ArangoDB server.

Parameters

- **resp** (arango.response.Response) – HTTP response.
- **msg** (*str*) – Error message override.

Variables

- **source** (*str*) – Source of the error (always set to "server").
- **message** (*str*) – Exception message.
- **url** (*str*) – API URL.

- **response** (arango.response.Response) – HTTP response object.
- **request** (arango.request.Request) – HTTP request object.
- **http_method** (*str*) – HTTP method in lowercase (e.g. "post").
- **http_code** (*int*) – HTTP status code.
- **http_headers** (*dict*) – Response headers.
- **error_code** (*int*) – Error code from ArangoDB server.
- **error_message** (*str*) – Raw error message from ArangoDB server.

exception arango.exceptions.**AQLQueryListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve running AQL queries.

exception arango.exceptions.**AQLQueryExplainError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to parse and explain query.

exception arango.exceptions.**AQLQueryValidateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to parse and validate query.

exception arango.exceptions.**AQLQueryExecuteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute query.

exception arango.exceptions.**AQLQueryKillError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to kill the query.

exception arango.exceptions.**AQLQueryClearError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to clear slow AQL queries.

exception arango.exceptions.**AQLQueryTrackingGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve AQL tracking properties.

exception arango.exceptions.**AQLQueryTrackingSetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to configure AQL tracking properties.

exception arango.exceptions.**AQLCachePropertiesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve query cache properties.

exception arango.exceptions.**AQLCacheConfigureError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to configure query cache properties.

exception arango.exceptions.**AQLCacheEntriesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve AQL cache entries.

exception arango.exceptions.**AQLCacheClearError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to clear the query cache.

exception arango.exceptions.**AQLFunctionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve AQL user functions.

exception arango.exceptions.**AQLFunctionCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create AQL user function.

exception arango.exceptions.**AQLFunctionDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete AQL user function.

exception arango.exceptions.**AsyncExecuteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute async API request.

exception arango.exceptions.**AsyncJobListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve async jobs.

exception arango.exceptions.**AsyncJobCancelError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to cancel async job.

exception arango.exceptions.**AsyncJobStatusError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve async job status.

exception arango.exceptions.**AsyncJobResultError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve async job result.

exception arango.exceptions.**AsyncJobClearError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to clear async job results.

exception arango.exceptions.**BackupCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create a backup.

exception arango.exceptions.**BackupDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete a backup.

exception arango.exceptions.**BackupDownloadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to download a backup from remote repository.

exception arango.exceptions.**BackupGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve backup details.

exception arango.exceptions.**BackupRestoreError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to restore from backup.

exception arango.exceptions.**BackupUploadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to upload a backup to remote repository.

exception arango.exceptions.**BatchStateError**(*msg: str*)

The batch object was in a bad state.

exception arango.exceptions.**BatchJobResultError**(*msg: str*)

Failed to retrieve batch job result.

exception arango.exceptions.**BatchExecuteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute batch API request.

exception arango.exceptions.**CollectionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collections.

exception arango.exceptions.**CollectionPropertiesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection properties.

exception arango.exceptions.**CollectionConfigureError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to configure collection properties.

exception arango.exceptions.**CollectionStatisticsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection statistics.

exception arango.exceptions.**CollectionRevisionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection revision.

exception arango.exceptions.**CollectionChecksumError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection checksum.

exception arango.exceptions.**CollectionCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create collection.

exception arango.exceptions.**CollectionDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete collection.

exception arango.exceptions.**CollectionRenameError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to rename collection.

exception arango.exceptions.**CollectionTruncateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to truncate collection.

exception arango.exceptions.**CollectionLoadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to load collection.

exception arango.exceptions.**CollectionUnloadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to unload collection.

exception arango.exceptions.**CollectionRecalculateCountError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to recalculate document count.

exception arango.exceptions.**CollectionResponsibleShardError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve responsible shard.

exception arango.exceptions.**CursorStateError**(*msg: str*)
The cursor object was in a bad state.

exception arango.exceptions.**CursorCountError**(*msg: str*)
The cursor count was not enabled.

exception arango.exceptions.**CursorEmptyError**(*msg: str*)
The current batch in cursor was empty.

exception arango.exceptions.**CursorNextError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve the next result batch from server.

exception arango.exceptions.**CursorCloseError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to delete the cursor result from server.

exception arango.exceptions.**DatabaseListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve databases.

exception arango.exceptions.**DatabasePropertiesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve database properties.

exception arango.exceptions.**DatabaseCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to create database.

exception arango.exceptions.**DatabaseDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to delete database.

exception arango.exceptions.**DocumentParseError**(*msg: str*)
Failed to parse document input.

exception arango.exceptions.**DocumentCountError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve document count.

exception arango.exceptions.**DocumentInError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to check whether document exists.

exception arango.exceptions.**DocumentGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve document.

exception arango.exceptions.**DocumentKeysError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)
Failed to retrieve document keys.

exception arango.exceptions.**DocumentIDsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve document IDs.

exception arango.exceptions.**DocumentInsertError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to insert document.

exception arango.exceptions.**DocumentReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace document.

exception arango.exceptions.**DocumentUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update document.

exception arango.exceptions.**DocumentDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete document.

exception arango.exceptions.**DocumentRevisionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

The expected and actual document revisions mismatched.

exception arango.exceptions.**FoxxServiceListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx services.

exception arango.exceptions.**FoxxServiceGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service metadata.

exception arango.exceptions.**FoxxServiceCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create Foxx service.

exception arango.exceptions.**FoxxServiceUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update Foxx service.

exception arango.exceptions.**FoxxServiceReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace Foxx service.

exception arango.exceptions.**FoxxServiceDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete Foxx services.

exception arango.exceptions.**FoxxConfigGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service configuration.

exception arango.exceptions.**FoxxConfigUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update Foxx service configuration.

exception arango.exceptions.**FoxxConfigReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace Foxx service configuration.

exception arango.exceptions.**FoxxDependencyGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service dependencies.

exception arango.exceptions.**FoxxDependencyUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update Foxx service dependencies.

exception arango.exceptions.**FoxxDependencyReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace Foxx service dependencies.

exception arango.exceptions.**FoxxScriptListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service scripts.

exception arango.exceptions.**FoxxScriptRunError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to run Foxx service script.

exception arango.exceptions.**FoxxTestRunError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to run Foxx service tests.

exception arango.exceptions.**FoxxDevModeEnableError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to enable development mode for Foxx service.

exception arango.exceptions.**FoxxDevModeDisableError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to disable development mode for Foxx service.

exception arango.exceptions.**FoxxReadmeGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service readme.

exception arango.exceptions.**FoxxSwaggerGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Foxx service swagger.

exception arango.exceptions.**FoxxDownloadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to download Foxx service bundle.

exception arango.exceptions.**FoxxCommitError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to commit local Foxx service state.

exception arango.exceptions.**GraphListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve graphs.

exception arango.exceptions.**GraphCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create the graph.

exception arango.exceptions.**GraphDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete the graph.

exception arango.exceptions.**GraphPropertiesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve graph properties.

exception arango.exceptions.**GraphTraverseError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute graph traversal.

exception arango.exceptions.**VertexCollectionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve vertex collections.

exception arango.exceptions.**VertexCollectionCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create vertex collection.

exception arango.exceptions.**VertexCollectionDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete vertex collection.

exception arango.exceptions.**EdgeDefinitionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve edge definitions.

exception arango.exceptions.**EdgeDefinitionCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create edge definition.

exception arango.exceptions.**EdgeDefinitionReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace edge definition.

exception arango.exceptions.**EdgeDefinitionDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete edge definition.

exception arango.exceptions.**EdgeListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve edges coming in and out of a vertex.

exception arango.exceptions.**IndexListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection indexes.

exception arango.exceptions.**IndexCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create collection index.

exception arango.exceptions.**IndexDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete collection index.

exception arango.exceptions.**IndexLoadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to load indexes into memory.

exception arango.exceptions.**PregelJobCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create Pregel job.

exception arango.exceptions.**PregelJobGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve Pregel job details.

exception arango.exceptions.**PregelJobDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete Pregel job.

exception arango.exceptions.**ServerConnectionError**(*msg: str*)

Failed to connect to ArangoDB server.

exception arango.exceptions.**ServerEngineError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve database engine.

exception arango.exceptions.**ServerVersionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server version.

exception arango.exceptions.**ServerDetailsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server details.

exception arango.exceptions.**ServerStatusError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server status.

exception arango.exceptions.**ServerTimeError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server system time.

exception arango.exceptions.**ServerEchoError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve details on last request.

exception arango.exceptions.**ServerShutdownError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to initiate shutdown sequence.

exception arango.exceptions.**ServerRunTestsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute server tests.

exception arango.exceptions.**ServerRequiredDBVersionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server target version.

exception arango.exceptions.**ServerReadLogError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve global log.

exception arango.exceptions.**ServerLogLevelError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server log levels.

exception arango.exceptions.**ServerLogLevelSetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to set server log levels.

exception arango.exceptions.**ServerReloadRoutingError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to reload routing details.

exception arango.exceptions.**ServerStatisticsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server statistics.

exception arango.exceptions.**ServerMetricsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server metrics.

exception arango.exceptions.**ServerRoleError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server role in a cluster.

exception arango.exceptions.**ServerTLSError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve TLS data.

exception arango.exceptions.**ServerTLSReloadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to reload TLS.

exception arango.exceptions.**ServerEncryptionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to reload user-defined encryption keys.

exception arango.exceptions.**TaskListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server tasks.

exception arango.exceptions.**TaskGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server task details.

exception arango.exceptions.**TaskCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create server task.

exception arango.exceptions.**TaskDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete server task.

exception arango.exceptions.**TransactionExecuteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to execute raw transaction.

exception arango.exceptions.**TransactionInitError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to initialize transaction.

exception arango.exceptions.**TransactionStatusError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve transaction status.

exception arango.exceptions.**TransactionCommitError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to commit transaction.

exception arango.exceptions.**TransactionAbortError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to abort transaction.

exception arango.exceptions.**UserListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve users.

exception arango.exceptions.**UserGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve user details.

exception arango.exceptions.**UserCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create user.

exception arango.exceptions.**UserUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update user.

exception arango.exceptions.**UserReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace user.

exception arango.exceptions.**UserDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete user.

exception arango.exceptions.**ViewListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve views.

exception arango.exceptions.**ViewGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve view details.

exception arango.exceptions.**ViewCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create view.

exception arango.exceptions.**ViewUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update view.

exception arango.exceptions.**ViewReplaceError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to replace view.

exception arango.exceptions.**ViewDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete view.

exception arango.exceptions.**ViewRenameError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to rename view.

exception arango.exceptions.**AnalyzerListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve analyzers.

exception arango.exceptions.**AnalyzerGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve analyzer details.

exception arango.exceptions.**AnalyzerCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create analyzer.

exception arango.exceptions.**AnalyzerDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete analyzer.

exception arango.exceptions.**PermissionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to list user permissions.

exception arango.exceptions.**PermissionGetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve user permission.

exception arango.exceptions.**PermissionUpdateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update user permission.

exception arango.exceptions.**PermissionResetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to reset user permission.

exception arango.exceptions.**WALPropertiesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve WAL properties.

exception arango.exceptions.**WALConfigureError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to configure WAL properties.

exception arango.exceptions.**WALTransactionListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve running WAL transactions.

exception arango.exceptions.**WALFlushError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to flush WAL.

exception arango.exceptions.**WALTickRangesError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to return WAL tick ranges.

exception arango.exceptions.**WALLastTickError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to return WAL tick ranges.

exception arango.exceptions.**WALTailError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to return WAL tick ranges.

exception arango.exceptions.**ReplicationInventoryError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve inventory of collection and indexes.

exception arango.exceptions.**ReplicationDumpBatchCreateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to create dump batch.

exception arango.exceptions.**ReplicationDumpBatchDeleteError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to delete a dump batch.

exception arango.exceptions.**ReplicationDumpBatchExtendError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to extend a dump batch.

exception arango.exceptions.**ReplicationDumpError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve collection content.

exception arango.exceptions.**ReplicationSyncError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to synchronize data from remote.

exception arango.exceptions.**ReplicationClusterInventoryError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve overview of collection and indexes in a cluster.

exception arango.exceptions.**ReplicationLoggerStateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve logger state.

exception arango.exceptions.**ReplicationLoggerFirstTickError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve logger first tick.

exception arango.exceptions.**ReplicationApplierConfigError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve replication applier configuration.

exception arango.exceptions.**ReplicationApplierConfigSetError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to update replication applier configuration.

exception arango.exceptions.**ReplicationApplierStartError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to start replication applier.

exception arango.exceptions.**ReplicationApplierStopError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to stop replication applier.

exception arango.exceptions.**ReplicationApplierStateError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve replication applier state.

exception arango.exceptions.**ReplicationMakeSlaveError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to change role to slave.

exception arango.exceptions.**ReplicationServerIDError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server ID.

exception arango.exceptions.**ClusterHealthError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve DBServer health.

exception arango.exceptions.**ClusterServerIDError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server ID.

exception arango.exceptions.**ClusterServerRoleError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server role.

exception arango.exceptions.**ClusterServerStatisticsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve DBServer statistics.

exception arango.exceptions.**ClusterServerVersionError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server node version.

exception arango.exceptions.**ClusterServerEngineError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve server node engine.

exception arango.exceptions.**ClusterMaintenanceModeError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to enable/disable cluster supervision maintenance mode.

exception arango.exceptions.**ClusterEndpointsError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve cluster endpoints.

exception arango.exceptions.**ClusterServerCountError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve cluster server count.

exception arango.exceptions.**JWTAuthError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to get a new JWT token from ArangoDB.

exception arango.exceptions.**JWTSecretListError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to retrieve information on currently loaded JWT secrets.

exception arango.exceptions.**JWTSecretReloadError**(*resp: arango.response.Response*, *request: arango.request.Request*, *msg: Optional[str] = None*)

Failed to reload JWT secrets.

3.23.4 Error Codes

The *errno* module contains a constant mapping to ArangoDB's error codes.

3.24 Logging

To see full HTTP request and response details, you can modify the logger settings for the Requests library, which python-arango uses under the hood:

```
import requests
import logging

try:
    # For Python 3
    from http.client import HTTPConnection
except ImportError:
    # For Python 2
    from httplib import HTTPConnection
HTTPConnection.debuglevel = 1

logging.basicConfig()
logging.getLogger().setLevel(logging.DEBUG)
requests_log = logging.getLogger("requests.packages.urllib3")
requests_log.setLevel(logging.DEBUG)
requests_log.propagate = True
```

If python-arango's default HTTP client is overridden, the code snippet above may not work as expected. See *HTTP Clients* for more information.

3.25 Authentication

Python-arango supports two HTTP authentication methods: basic and JSON Web Tokens (JWT).

3.25.1 Basic Authentication

This is python-arango's default authentication method.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user using basic auth.
db = client.db('test', username='root', password='passwd')

# The authentication method can be given explicitly.
```

(continues on next page)

(continued from previous page)

```
db = client.db(
    'test',
    username='root',
    password='passwd',
    auth_method='basic'
)
```

3.25.2 JSON Web Tokens (JWT)

Python-arango automatically obtains JSON web tokens from the server using username and password. It also refreshes expired tokens and retries requests. The client and server clocks must be synchronized for the automatic refresh to work correctly.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user using JWT.
db = client.db(
    'test',
    username='root',
    password='passwd',
    auth_method='jwt'
)

# Manually refresh the token.
db.conn.refresh_token()

# Override the token expiry compare leeway in seconds (default: 0) to
# compensate for out-of-sync clocks between the client and server.
db.conn.ext_leeway = 2
```

User generated JWT token can be used for superuser access.

Example:

```
from calendar import timegm
from datetime import datetime

import jwt

from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Generate the JWT token manually.
now = timegm(datetime.utcnow().utctimetuple())
token = jwt.encode(
    payload={
        'iat': now,
        'exp': now + 3600,
        'iss': 'arangodb',
        'server_id': 'client'
```

(continues on next page)

(continued from previous page)

```
    },
    key='secret',
).decode('utf-8')

# Connect to "test" database as superuser using the token.
db = client.db('test', superuser_token=token)
```

3.26 HTTP Clients

Python-arango lets you define your own HTTP client for sending requests to ArangoDB server. The default implementation uses the requests library.

Your HTTP client must inherit *arango.http.HTTPClient* and implement the following abstract methods:

- *arango.http.HTTPClient.create_session()*
- *arango.http.HTTPClient.send_request()*

The **create_session** method must return a requests.Session instance per connected host (coordinator). The session objects are stored in the client.

The **send_request** method must use the session to send an HTTP request, and return a fully populated instance of *arango.response.Response*.

For example, let's say your HTTP client needs:

- Automatic retries
- Additional HTTP header called `x-my-header`
- SSL certificate verification disabled
- Custom logging

Your `CustomHTTPClient` class might look something like this:

```
import logging

from requests.adapters import HTTPAdapter
from requests import Session
from requests.packages.urllib3.util.retry import Retry

from arango.response import Response
from arango.http import HTTPClient


class CustomHTTPClient(HTTPClient):
    """My custom HTTP client with cool features."""

    def __init__(self):
        # Initialize your logger.
        self._logger = logging.getLogger('my_logger')

    def create_session(self, host):
        session = Session()

        # Add request header.
        session.headers.update({'x-my-header': 'true'})
```

(continues on next page)

(continued from previous page)

```
        # Enable retries.
        retry_strategy = Retry(
            total=3,
            backoff_factor=1,
            status_forcelist=[429, 500, 502, 503, 504],
            method_whitelist=["HEAD", "GET", "OPTIONS"],
        )
        http_adapter = HTTPAdapter(max_retries=retry_strategy)
        session.mount('https://', adapter)
        session.mount('http://', adapter)

        return session

    def send_request(self,
                     session,
                     method,
                     url,
                     params=None,
                     data=None,
                     headers=None,
                     auth=None):
        # Add your own debug statement.
        self._logger.debug(f'Sending request to {url}')

        # Send a request.
        response = session.request(
            method=method,
            url=url,
            params=params,
            data=data,
            headers=headers,
            auth=auth,
            verify=False,  # Disable SSL verification
            timeout=5      # Use timeout of 5 seconds
        )
        self._logger.debug(f'Got {response.status_code}')

        # Return an instance of arango.response.Response.
        return Response(
            method=response.request.method,
            url=response.url,
            headers=response.headers,
            status_code=response.status_code,
            status_text=response.reason,
            raw_body=response.text,
        )
```

Then you would inject your client as follows:

```
from arango import ArangoClient

from my_module import CustomHTTPClient

client = ArangoClient(
    hosts='http://localhost:8529',
    http_client=CustomHTTPClient()
```

(continues on next page)

(continued from previous page)

```
)
```

See requests.Session for more details on how to create and manage sessions.

3.27 Replication

Replication allows you to replicate data onto another machine. It forms the basis of all disaster recovery and failover features ArangoDB offers. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the Replication API wrapper.
replication = db.replication

# Create a new dump batch.
batch = replication.create_dump_batch(ttl=1000)

# Extend an existing dump batch.
replication.extend_dump_batch(batch['id'], ttl=1000)

# Get an overview of collections and indexes.
replication.inventory(
    batch_id=batch['id'],
    include_system=True,
    all_databases=True
)

# Get an overview of collections and indexes in a cluster.
replication.cluster_inventory(include_system=True)

# Get the events data for given collection.
replication.dump(
    collection='students',
    batch_id=batch['id'],
    lower=0,
    upper=1000000,
    chunk_size=0,
    include_system=True,
    ticks=0,
    flush=True,
)

# Delete an existing dump batch.
replication.delete_dump_batch(batch['id'])

# Get the logger state.
replication.logger_state()
```

(continues on next page)

(continued from previous page)

```
# Get the logger first tick value.
replication.logger_first_tick()

# Get the replication applier configuration.
replication.applier_config()

# Update the replication applier configuration.
result = replication.set_applier_config(
    endpoint='http://127.0.0.1:8529',
    database='test',
    username='root',
    password='passwd',
    max_connect_retries=120,
    connect_timeout=15,
    request_timeout=615,
    chunk_size=0,
    auto_start=True,
    adaptive_polling=False,
    include_system=True,
    auto_resync=True,
    auto_resync_retries=3,
    initial_sync_max_wait_time=405,
    connection_retry_wait_time=25,
    idle_min_wait_time=2,
    idle_max_wait_time=3,
    require_from_present=False,
    verbose=True,
    restrict_type='include',
    restrict_collections=['students']
)

# Get the replication applier state.
replication.applier_state()

# Start the replication applier.
replication.start_applier()

# Stop the replication applier.
replication.stop_applier()

# Get the server ID.
replication.server_id()

# Synchronize data from a remote (master) endpoint
replication.synchronize(
    endpoint='tcp://master:8500',
    database='test',
    username='root',
    password='passwd',
    include_system=False,
    incremental=False,
    restrict_type='include',
    restrict_collections=['students']
)
```

See *Replication* for API specification.

3.28 Clusters

Python-arango provides APIs for working with ArangoDB clusters. For more information on the design and architecture, refer to ArangoDB manual.

3.28.1 Coordinators

To connect to multiple ArangoDB coordinators, you must provide either a list of host strings or a comma-separated string during client initialization.

Example:

```
from arango import ArangoClient

# Single host
client = ArangoClient(hosts='http://localhost:8529')

# Multiple hosts (option 1: list)
client = ArangoClient(hosts=['http://host1:8529', 'http://host2:8529'])

# Multiple hosts (option 2: comma-separated string)
client = ArangoClient(hosts='http://host1:8529,http://host2:8529')
```

By default, a requests.Session instance is created per coordinator. HTTP requests to a host are sent using only its corresponding session. For more information on how to override this behaviour, see *HTTP Clients*.

3.28.2 Load-Balancing Strategies

There are two load-balancing strategies available: "roundrobin" and "random" (defaults to "roundrobin" if unspecified).

Example:

```
from arango import ArangoClient

hosts = ['http://host1:8529', 'http://host2:8529']

# Round-robin
client = ArangoClient(hosts=hosts, host_resolver='roundrobin')

# Random
client = ArangoClient(hosts=hosts, host_resolver='random')
```

3.28.3 Administration

Below is an example on how to manage clusters using python-arango.

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
```

(continues on next page)

(continued from previous page)

```
sys_db = client.db('_system', username='root', password='passwd')

# Get the Cluster API wrapper.
cluster = sys_db.cluster

# Get this server's ID.
cluster.server_id()

# Get this server's role.
cluster.server_role()

# Get the cluster health.
cluster.health()

# Get cluster server details.
cluster.server_count()
server_id = cluster.server_id()
cluster.server_engine(server_id)
cluster.server_version(server_id)
cluster.server_statistics(server_id)

# Toggle maintenance mode (allowed values are "on" and "off").
cluster.toggle_maintenance_mode('on')
cluster.toggle_maintenance_mode('off')
```

See *ArangoClient* and *Cluster* for API specification.

3.29 JSON Serialization

You can provide your own JSON serializer and deserializer during client initialization. They must be callables that take a single argument.

Example:

```
import json

from arango import ArangoClient

# Initialize the ArangoDB client with custom serializer and deserializer.
client = ArangoClient(
    hosts='http://localhost:8529',
    serializer=json.dumps,
    deserializer=json.loads
)
```

See *ArangoClient* for API specification.

3.30 Backups

Backups are near consistent snapshots of an entire ArangoDB service including databases, collections, indexes, views and users. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db(
    '_system',
    username='root',
    password='passwd',
    auth_method='jwt'
)

# Get the backup API wrapper.
backup = sys_db.backup

# Create a backup.
result = backup.create(
    label='foo',
    allow_inconsistent=True,
    force=False,
    timeout=1000
)
backup_id = result['backup_id']

# Retrieve details on all backups
backup.get()

# Retrieve details on a specific backup.
backup.get(backup_id=backup_id)

# Upload a backup to a remote repository.
result = backup.upload(
    backup_id=backup_id,
    repository='local://tmp/backups',
    config={'local': {'type': 'local'}}
)
upload_id = result['upload_id']

# Get status of an upload.
backup.upload(upload_id=upload_id)

# Abort an upload.
backup.upload(upload_id=upload_id, abort=True)

# Download a backup from a remote repository.
result = backup.download(
    backup_id=backup_id,
    repository='local://tmp/backups',
    config={'local': {'type': 'local'}}
)
download_id = result['download_id']

# Get status of an download.
backup.download(download_id=download_id)

# Abort an download.
```

(continues on next page)

(continued from previous page)

```
backup.download(download_id=download_id, abort=True)

# Restore from a backup.
backup.restore(backup_id)

# Delete a backup.
backup.delete(backup_id)
```

See *Backup* for API specification.

3.31 Error Codes

Python-arango provides ArangoDB error code constants for convenience.

Example

```
from arango import errno

# Some examples
assert errno.NOT_IMPLEMENTED == 9
assert errno.DOCUMENT_REV_BAD == 1239
assert errno.DOCUMENT_NOT_FOUND == 1202
```

For more information, refer to ArangoDB manual.

3.32 Contributing

3.32.1 Requirements

Before submitting a pull request on GitHub, please make sure you meet the following requirements:

- The pull request points to dev branch.
- Changes are squashed into a single commit. I like to use git rebase for this.
- Commit message is in present tense. For example, "Fix bug" is good while "Fixed bug" is not.
- Sphinx-compatible docstrings.
- PEP8 compliance.
- No missing docstrings or commented-out lines.
- Test coverage remains at %100. If a piece of code is trivial and does not need unit tests, use this to exclude it from coverage.
- No build failures on Travis CI. Builds automatically trigger on pull request submissions.
- Documentation is kept up-to-date with the new changes (see below).

Warning: The dev branch is occasionally rebased, and its commit history may be overwritten in the process. Before you begin your feature work, git fetch or pull to ensure that your local branch has not diverged. If you see git conflicts and want to start with a clean slate, run the following commands:

```
~$ git checkout dev
~$ git fetch origin
~$ git reset --hard origin/dev  # THIS WILL WIPE ALL LOCAL CHANGES
```

3.32.2 Style

To ensure PEP8 compliance, run flake8:

```
~$ pip install flake8
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ flake8
```

If there is a good reason to ignore a warning, see here on how to exclude it.

3.32.3 Testing

To test your changes, you can run the integration test suite that comes with **python-arango**. It uses pytest and requires an actual ArangoDB instance.

To run the test suite (use your own host, port and root password):

```
~$ pip install pytest
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ py.test --complete --host=127.0.0.1 --port=8529 --passwd=passwd
```

To run the test suite with coverage report:

```
~$ pip install coverage pytest pytest-cov
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ py.test --complete --host=127.0.0.1 --port=8529 --passwd=passwd --cov=kq
```

As the test suite creates real databases and jobs, it should only be run in development environments.

3.32.4 Documentation

The documentation including the README is written in reStructuredText and uses Sphinx. To build an HTML version on your local machine:

```
~$ pip install sphinx sphinx_rtd_theme
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango/docs
~$ sphinx-build . build  # Open build/index.html in a browser
```

As always, thank you for your contribution!

3.33 API Specification

This page contains the specification for all classes and methods available in python-arango.

3.33.1 ArangoClient

class arango.client.**ArangoClient**(*hosts: Union[str, Sequence[str]] = 'http://127.0.0.1:8529', host_resolver: str = 'roundrobin', http_client: Optional[arango.http.HTTPClient] = None, serializer: Callable[[...], str] = <function ArangoClient.<lambda>>, deserializer: Callable[[str], Any] = <function ArangoClient.<lambda>>*)

ArangoDB client.

Parameters

- **hosts** (*str | [str]*) – Host URL or list of URLs (coordinators in a cluster).
- **host_resolver** (*str*) – Host resolver. This parameter used for clusters (when multiple host URLs are provided). Accepted values are "roundrobin" and "random". Any other value defaults to round robin.
- **http_client** (arango.http.HTTPClient) – User-defined HTTP client.
- **serializer** (*callable*) – User-defined JSON serializer. Must be a callable which takes a JSON data type object as its only argument and return the serialized string. If not given, `json.dumps` is used by default.
- **deserializer** (*callable*) – User-defined JSON de-serializer. Must be a callable which takes a JSON serialized string as its only argument and return the de-serialized object. If not given, `json.loads` is used by default.

close() → None

Close HTTP sessions.

hosts

Return the list of ArangoDB host URLs.

Returns List of ArangoDB host URLs.

Return type [str]

version

Return the client version.

Returns Client version.

Return type str

db(*name: str = '_system', username: str = 'root', password: str = '', verify: bool = False, auth_method: str = 'basic', superuser_token: Optional[str] = None*) → arango.database.StandardDatabase

Connect to an ArangoDB database and return the database API wrapper.

Parameters

- **name** (*str*) – Database name.
- **username** (*str*) – Username for basic authentication.
- **password** (*str*) – Password for basic authentication.
- **verify** (*bool*) – Verify the connection by sending a test request.
- **auth_method** (*str*) – HTTP authentication method. Accepted values are "basic" (default) and "jwt". If set to "jwt", the token is refreshed automatically using ArangoDB username and password. This assumes that the clocks of the server and client are synchronized.

- **superuser_token** (*str*) – User generated token for superuser access. If set, parameters **username**, **password** and **auth_method** are ignored. This token is not refreshed automatically.

Returns Standard database API wrapper.

Return type *arango.database.StandardDatabase*

Raises **arango.exceptions.ServerConnectionError** – If **verify** was set to True and the connection fails.

3.33.2 AsyncDatabase

class arango.database.**AsyncDatabase**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], return_result: bool*)

Database API wrapper tailored specifically for async execution.

See *arango.database.StandardDatabase.begin_async_execution()*.

Parameters

- **connection** – HTTP connection.
- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.AsyncJob*, which you can use to retrieve results from server once available. If set to False, API executions return None and no results are stored on server.

analyzer(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return analyzer details.

Parameters **name** (*str*) – Analyzer name.

Returns Analyzer details.

Return type dict

Raises **arango.exceptions.AnalyzerGetError** – If retrieval fails.

analyzers() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

Return list of analyzers.

Returns List of analyzers.

Return type [dict]

Raises **arango.exceptions.AnalyzerListError** – If retrieval fails.

aql

Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status: str, count: Optional[int] = None*) → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return IDs of async jobs with given status.

Parameters

- **status** (*str*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str]

Raises ***arango.exceptions.AsyncJobListError*** – If retrieval fails.

backup
Return Backup API wrapper.

Returns Backup API wrapper.

Return type *arango.backup.Backup*

clear_async_jobs(*threshold: Optional[int] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int | None*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

cluster
Return Cluster API wrapper.

Returns Cluster API wrapper.

Return type *arango.cluster.Cluster*

collection(*name: str*) → arango.collection.StandardCollection
Return the standard collection API wrapper.

Parameters **name** (*str*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

conn
Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str

create_analyzer(*name: str*, *analyzer_type: str*, *properties: Optional[Dict[str, Any]] = None*, *features: Optional[Sequence[str]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **analyzer_type** (*str*) – Analyzer type.
- **properties** (*dict | None*) – Analyzer properties.
- **features** (*list | None*) – Analyzer features.

Returns Analyzer details.

Return type dict

Raises ***arango.exceptions.AnalyzerCreateError*** – If create fails.

create_arangosearch_view(*name: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict | None*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewCreateError*** – If create fails.

create_collection(*name: str*, *sync: bool = False*, *system: bool = False*, *edge: bool = False*, *user_keys: bool = True*, *key_increment: Optional[int] = None*, *key_offset: Optional[int] = None*, *key_generator: str = 'traditional'*, *shard_fields: Optional[Sequence[str]] = None*, *shard_count: Optional[int] = None*, *replication_factor: Optional[int] = None*, *shard_like: Optional[str] = None*, *sync_replication: Optional[bool] = None*, *enforce_replication_factor: Optional[bool] = None*, *sharding_strategy: Optional[str] = None*, *smart_join_attribute: Optional[str] = None*, *write_concern: Optional[int] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[arango.collection.StandardCollection, arango.job.AsyncJob[arango.collection.StandardCollection][arango.collection.StandardCollection], arango.job.BatchJob[arango.collection.StandardCollection][arango.collection.StandardCollection], None]
Create a new collection.

Parameters

- **name** (*str*) – Collection name.
- **sync** (*bool | None*) – If set to True, document operations via the collection will block until synchronized to disk by default.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **key_generator** (*str*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.
- **sharding_strategy** (*str*) – Sharding strategy. Available for ArangoDB version and up only. Possible values are "community-compat", "enterprise-compat", "enterprise-smart-edge-compat", "hash" and "enterprise-hash-smart-edge". Refer to ArangoDB documentation for more details on each value.
- **smart_join_attribute** (*str*) – Attribute of the collection which must contain the shard key value of the smart join collection. The shard key for the documents must contain the value of this attribute, followed by a colon ":" and the primary key of the document. Requires parameter **shard_like** to be set to the name of another collection, and parameter **shard_fields** to be set to a single shard key attribute, with another colon ":" at the end. Available only for enterprise version of ArangoDB.
- **write_concern** (*int*) – Write concern for the collection. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time. The value of this parameter cannot be larger than that of **replication_factor**. Default value is 1. Used for clusters only.

- **schema** (*dict*) – Optional dict specifying the collection level schema for documents. See ArangoDB documentation for more information on document schema validation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name: str*, *users: Optional[Sequence[Dict[str, Any]]] = None*, *replication_factor: Union[int, str, None] = None*, *write_concern: Optional[int] = None*, *sharding: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Create a new database.

Parameters

- **name** (*str*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.
- **replication_factor** (*int | str*) – Default replication factor for collections created in this database. Special values include "satellite" which replicates the collection to every DBServer, and 1 which disables replication. Used for clusters only.
- **write_concern** (*int*) – Default write concern for collections created in this database. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time, however. Value of this parameter can not be larger than the value of **replication_factor**. Used for clusters only.
- **sharding** (*str*) – Sharding method used for new collections in this database. Allowed values are: "", "flexible" and "single". The first two are equivalent. Used for clusters only.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name: str*, *edge_definitions: Optional[Sequence[Dict[str, Any]]] = None*, *orphan_collections: Optional[Sequence[str]] = None*, *smart: Optional[bool] = None*, *smart_field: Optional[str] = None*, *shard_count: Optional[int] = None*) → Union[arango.graph.Graph, arango.job.AsyncJob[arango.graph.Graph][arango.graph.Graph], arango.job.BatchJob[arango.graph.Graph][arango.graph.Graph], None]

Create a new graph.

Parameters

- **name** (*str*) – Graph name.

- **edge_definitions** (*[dict] | None*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str] | None*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool | None*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | None*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int | None*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name: str*, *command: str*, *params: Optional[Dict[str, Any]] = None*, *period: Optional[int] = None*, *offset: Optional[int] = None*, *task_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new server task.

Parameters

- **name** (*str*) – Name of the server task.
- **command** (*str*) – Javascript command to execute.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **period** (*int | None*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int | None*) – Initial delay before execution in seconds.
- **task_id** (*str | None*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises ***arango.exceptions.TaskCreateError*** – If create fails.

create_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – Password.
- **active** (*bool | None*) – True if user is active, False otherwise.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserCreateError*** – If create fails.

create_view(*name: str*, *view_type: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a view.

Parameters

- **name** (*str*) – View name.
- **view_type** (*str*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewCreateError*** – If create fails.

databases() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return the names all databases.

Returns Database names.

Return type [str]

Raises ***arango.exceptions.DatabaseListError*** – If retrieval fails.

db_name

Return the name of the current database.

Returns Database name.

Return type str

delete_analyzer(*name: str*, *force: bool = False*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **force** (*bool*) – Remove the analyzer configuration even if in use.
- **ignore_missing** (*bool*) – Do not raise an exception on missing analyzer.

Returns True if analyzer was deleted successfully, False if analyzer was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.AnalyzerDeleteError* – If delete fails.

delete_collection(*name: str*, *ignore_missing: bool = False*, *system: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete the collection.

Parameters

- **name** (*str*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete the database.

Parameters

- **name** (*str*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document: Union[str, Dict[str, Any]]*, *rev: Optional[str] = None*, *check_rev: bool = True*, *ignore_missing: bool = False*, *return_old: bool = False*, *sync: Optional[bool] = None*, *silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name: str*, *ignore_missing: bool = False*, *drop_collections: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Drop the graph of the given name from the database.

Parameters

- **name** (*str*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool | None*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a server task.

Parameters

- **task_id** (*str*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a user.

Parameters

- **username** (*str*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.UserDeleteError* – If delete fails.

delete_view(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete a view.

Parameters

- **name** (*str*) – View name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing view.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document: Dict[str, Any]*, *rev: Optional[str] = None*, *check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]
Return a document.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- *arango.exceptions.DocumentGetError* – If retrieval fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

echo() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises *arango.exceptions.ServerEchoError* – If retrieval fails.

encryption() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Rotate the user-supplied keys for encryption.

This method is available only for enterprise edition of ArangoDB.

Returns New TLS data.

Return type dict

Raises ***arango.exceptions.ServerEncryptionError*** – If retrieval fails.

engine() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the database engine details.

Returns Database engine details.

Return type dict

Raises ***arango.exceptions.ServerEngineError*** – If retrieval fails.

execute_transaction(*command: str*, *params: Optional[Dict[str, Any]] = None*, *read: Optional[Sequence[str]] = None*, *write: Optional[Sequence[str]] = None*, *sync: Optional[bool] = None*, *timeout: Optional[numbers.Number] = None*, *max_size: Optional[int] = None*, *allow_implicit: Optional[bool] = None*, *intermediate_commit_count: Optional[int] = None*, *intermediate_commit_size: Optional[int] = None*) → Union[Any, arango.job.AsyncJob[typing.Any][Any], arango.job.BatchJob[typing.Any][Any], None]
Execute raw Javascript command in transaction.

Parameters

- **command** (*str*) – Javascript command to execute.
- **read** (*[str] | None*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str] | None*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **timeout** (*int | None*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int | None*) – Max transaction size limit in bytes.
- **allow_implicit** (*bool | None*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int | None*) – Max number of operations after which an intermediate commit is performed automatically.
- **intermediate_commit_size** (*int | None*) – Max size of operations in bytes after which an intermediate commit is performed automatically.

Returns Return value of **command**.

Return type Any

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name: str*) → arango.graph.Graph
Return the graph API wrapper.

Parameters **name** (*str*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]]][List[Dict[str, Any]]], None]
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if collection exists in the database.

Parameters **name** (*str*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a database exists.

Parameters **name** (*str*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document: Dict[str, Any], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a document exists.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a graph exists in the database.

Parameters **name** (*str*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if user exists.

Parameters **username** (*str*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection: str, document: Dict[str, Any], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False, overwrite_mode: Optional[str] = None, keep_none: Optional[bool] = None, merge: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Insert a new document.

Parameters

- **collection** (*str*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.
- **overwrite_mode** (*str | None*) – Overwrite behavior used when the document key exists already. Allowed values are "replace" (replace-insert) or "update" (update-insert). Implicitly sets the value of parameter **overwrite**.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely. Applies only when **overwrite_mode** is set to "update" (update-insert).

- **merge** (*bool | None*) – If set to True (default), sub-dictionaries are merged instead of the new one overwriting the old one. Applies only when **overwrite_mode** is set to "update" (update-insert).

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return information on currently loaded JWT secrets.

Returns Information on currently loaded JWT secrets.

Return type dict

log_levels() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return current logging levels.

Returns Current logging levels.

Return type dict

metrics() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server metrics in Prometheus format.

Returns Server metrics in Prometheus format.

Return type str

name
Return database name.

Returns Database name.

Return type str

permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns Permission for given database or collection.

Return type str

Raises ***arango.exceptions.PermissionGetError*** – If retrieval fails.

permissions(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user permissions for all databases and collections.

Parameters **username** (*str*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raises *arango.exceptions.PermissionListError* – If retrieval fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return database properties.

Returns Database properties.

Return type dict

Raises *arango.exceptions.DatabasePropertiesError* – If retrieval fails.

read_log(*upto: Union[str, int, None] = None*, *level: Union[str, int, None] = None*, *start: Optional[int] = None*, *size: Optional[int] = None*, *offset: Optional[int] = None*, *search: Optional[str] = None*, *sort: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Read the global log from server.

Parameters

- **upto** (*int | str*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*int | str*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str*) – Return only the log entries containing the given text.
- **sort** (*str*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises *arango.exceptions.ServerReadLogError* – If read fails.

reload_jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Hot-reload JWT secrets.

Calling this without payload reloads JWT secrets from disk. Only files specified via arangod startup option `--server.jwt-secret-keyfile` or `--server.jwt-secret-folder` are used. It is not possible to change the location where files are loaded from without restarting the server.

Returns Information on reloaded JWT secrets.

Return type dict

reload_routing() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises ***arango.exceptions.ServerReloadRoutingError*** – If reload fails.

reload_tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Reload TLS data (server key, client-auth CA).

Returns New TLS data.

Return type dict

rename_view(*name: str*, *new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Rename a view.

Parameters

- **name** (*str*) – View name.
- **new_name** (*str*) – New view name.

Returns True if view was renamed successfully.

Return type bool

Raises ***arango.exceptions.ViewRenameError*** – If delete fails.

replace_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Replace an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replace_document(*document: Dict[str, Any], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.

- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username: str*, *password: str*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replication

Return Replication API wrapper.

Returns Replication API wrapper.

Return type *arango.replication.Replication*

required_db_version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return required version of target database.

Returns Required version of target database.

Return type str

Raises ***arango.exceptions.ServerRequiredDBVersionError*** – If retrieval fails.

reset_permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reset user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server role.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY", "AGENT" (Agency node in a cluster) or "UNDEFINED".

Return type str

Raises ***arango.exceptions.ServerRoleError*** – If retrieval fails.

run_tests(*tests: Sequence[str]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Run available unittests on the server.

Parameters **tests** (*[str]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises ***arango.exceptions.ServerRunTestsError*** – If execution fails.

set_log_levels(***kwargs*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
```

(continues on next page)

(continued from previous page)

```
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises ***arango.exceptions.ServerShutdownError*** – If shutdown fails.

statistics(*description: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return server statistics.

Returns Server statistics.

Return type dict

Raises ***arango.exceptions.ServerStatisticsError*** – If retrieval fails.

status() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises ***arango.exceptions.ServerStatusError*** – If retrieval fails.

task(*task_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the details of an active server task.

Parameters **task_id** (*str*) – Server task ID.

Returns Server task details.

Return type dict

Raises ***arango.exceptions.TaskGetError*** – If retrieval fails.

tasks() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises ***arango.exceptions.TaskListError*** – If retrieval fails.

time() → Union[datetime.datetime, arango.job.AsyncJob[datetime.datetime][datetime.datetime], arango.job.BatchJob[datetime.datetime][datetime.datetime], None]
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return TLS data (server key, client-auth CA).

Returns TLS data.

Return type dict

update_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

update_document(*document: Dict[str, Any], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username: str*, *permission: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Update user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **permission** (*str*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name: str*, *properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user details.

Parameters **username** (*str*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str

users() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return ArangoDB server version.

Returns Server version.

Return type str

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of views and their summaries.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.33.3 AsyncJob

class arango.job.**AsyncJob**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], job_id: str, response_handler: Callable[[arango.response.Response], T]*)

Job for tracking and retrieving result of an async API execution.

Parameters

- **connection** – HTTP connection.
- **job_id** (*str*) – Async job ID.
- **response_handler** (*callable*) – HTTP response handler.

id

Return the async job ID.

Returns Async job ID.

Return type str

status() → str

Return the async job status from server.

Once a job result is retrieved via func:*arango.job.AsyncJob.result* method, it is deleted from server and subsequent status queries will fail.

Returns Async job status. Possible values are "pending" (job is still in queue), "done" (job finished or raised an error), or "cancelled" (job was cancelled before completion).

Return type str

Raises ***arango.exceptions.AsyncJobStatusError*** – If retrieval fails.

result() → T

Return the async job result from server.

If the job raised an exception, it is propagated up at this point.

Once job result is retrieved, it is deleted from server and subsequent queries for result will fail.

Returns Async job result.

Raises

- ***arango.exceptions.ArangoError*** – If the job raised an exception.
- ***arango.exceptions.AsyncJobResultError*** – If retrieval fails.

cancel(*ignore_missing: bool = False*) → bool

Cancel the async job.

An async job cannot be cancelled once it is taken out of the queue.

Parameters **ignore_missing** (*bool*) – Do not raise an exception on missing job.

Returns True if job was cancelled successfully, False if the job was not found but **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.AsyncJobCancelError*** – If cancel fails.

clear(*ignore_missing: bool = False*) → bool
Delete the job result from the server.

Parameters **ignore_missing** (*bool*) – Do not raise an exception on missing job.

Returns True if result was deleted successfully, False if the job was not found but **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.AsyncJobClearError* – If delete fails.

3.33.4 AQL

class arango.aql.**AQL**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)
AQL (ArangoDB Query Language) API wrapper.

Parameters

- **connection** – HTTP connection.
- **executor** – API executor.

cache
Return the query cache API wrapper.

Returns Query cache API wrapper.

Return type *arango.aql.AQLQueryCache*

explain(*query: str*, *all_plans: bool = False*, *max_plans: Optional[int] = None*, *opt_rules: Optional[Sequence[str]] = None*, *bind_vars: Optional[MutableMapping[str, str]] = None*) → Union[Dict[str, Any], List[Dict[str, Any]], arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], typing.List[typing.Dict[str, typing.Any]]]][Union[Dict[str, Any], List[Dict[str, Any]]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], typing.List[typing.Dict[str, typing.Any]]]][Union[Dict[str, Any], List[Dict[str, Any]]]], None]
Inspect the query and return its metadata without executing it.

Parameters

- **query** (*str*) – Query to inspect.
- **all_plans** (*bool*) – If set to True, all possible execution plans are returned in the result. If set to False, only the optimal plan is returned.
- **max_plans** (*int*) – Total number of plans generated by the optimizer.
- **opt_rules** (*list*) – List of optimizer rules.
- **bind_vars** (*dict*) – Bind variables for the query.

Returns Execution plan, or plans if **all_plans** was set to True.

Return type dict | list

Raises *arango.exceptions.AQLQueryExplainError* – If explain fails.

validate(*query: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Parse and validate the query without executing it.

Parameters **query** (*str*) – Query to validate.

Returns Query details.

Return type dict

Raises *arango.exceptions.AQLQueryValidateError* – If validation fails.

execute(*query: str*, *count: bool = False*, *batch_size: Optional[int] = None*, *ttl: Optional[numbers.Number] = None*, *bind_vars: Optional[MutableMapping[str, str]] = None*, *full_count: Optional[bool] = None*, *max_plans: Optional[int] = None*, *optimizer_rules: Optional[Sequence[str]] = None*, *cache: Optional[bool] = None*, *memory_limit: int = 0*, *fail_on_warning: Optional[bool] = None*, *profile: Optional[bool] = None*, *max_transaction_size: Optional[int] = None*, *max_warning_count: Optional[int] = None*, *intermediate_commit_count: Optional[int] = None*, *intermediate_commit_size: Optional[int] = None*, *satellite_sync_wait: Optional[int] = None*, *stream: Optional[bool] = None*, *skip_inaccessible_cols: Optional[bool] = None*, *max_runtime: Optional[numbers.Number] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Execute the query and return the result cursor.

Parameters

- **query** (*str*) – Query to execute.
- **count** (*bool*) – If set to True, the total document count is included in the result cursor.
- **batch_size** (*int*) – Number of documents fetched by the cursor in one round trip.
- **ttl** (*int*) – Server side time-to-live for the cursor in seconds.
- **bind_vars** (*dict*) – Bind variables for the query.
- **full_count** (*bool*) – This parameter applies only to queries with LIMIT clauses. If set to True, the number of matched documents before the last LIMIT clause executed is included in the cursor. This is similar to MySQL SQL_CALC_FOUND_ROWS hint. Using this disables a few LIMIT optimizations and may lead to a longer query execution.
- **max_plans** (*int*) – Max number of plans the optimizer generates.
- **optimizer_rules** (*[str]*) – List of optimizer rules.
- **cache** (*bool*) – If set to True, the query cache is used. The operation mode of the query cache must be set to "on" or "demand".
- **memory_limit** (*int*) – Max amount of memory the query is allowed to use in bytes. If the query goes over the limit, it fails with error "resource limit exceeded". Value 0 indicates no limit.
- **fail_on_warning** (*bool*) – If set to True, the query throws an exception instead of producing a warning. This parameter can be used during development to catch issues early. If set to False, warnings are returned with the query result. There is a server configuration option "–query.fail-on-warning" for setting the default value for this behaviour so it does not need to be set per-query.
- **profile** (*bool*) – Return additional profiling details in the cursor, unless the query cache is used.
- **max_transaction_size** (*int*) – Transaction size limit in bytes.
- **max_warning_count** (*int*) – Max number of warnings returned.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically.

- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically.
- **satellite_sync_wait** (*int | float*) – Number of seconds in which the server must synchronize the satellite collections involved in the query. When the threshold is reached, the query is stopped. Available only for enterprise version of ArangoDB.
- **stream** (*bool*) – If set to True, query is executed in streaming fashion: query result is not stored server-side but calculated on the fly. Note: long-running queries hold collection locks for as long as the cursor exists. If set to False, query is executed right away in its entirety. Results are either returned right away (if the result set is small enough), or stored server-side and accessible via cursors (while respecting the ttl). You should use this parameter only for short-running queries or without exclusive locks. Note: parameters **cache**, **count** and **full_count** do not work for streaming queries. Query statistics, warnings and profiling data are made available only after the query is finished. Default value is False.
- **skip_inaccessible_cols** (*bool*) – If set to True, collections without user access are skipped, and query executes normally instead of raising an error. This helps certain use cases: a graph may contain several collections, and users with different access levels may execute the same query. This parameter lets you limit the result set by user access. Cannot be used in *transactions* and is available only for enterprise version of ArangoDB. Default value is False.
- **max_runtime** (*int | float*) – Query must be executed within this given timeout or it is killed. The value is specified in seconds. Default value is 0.0 (no timeout).

Returns Result cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.AQLQueryExecuteError*** – If execute fails.

kill(*query_id: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Kill a running query.

Parameters **query_id** (*str*) – Query ID.

Returns True if kill request was sent successfully.

Return type bool

Raises ***arango.exceptions.AQLQueryKillError*** – If the send fails.

queries() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

Return the currently running AQL queries.

Returns Running AQL queries.

Return type [dict]

Raises ***arango.exceptions.AQLQueryListError*** – If retrieval fails.

slow_queries() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

Return a list of all slow AQL queries.

Returns Slow AQL queries.

Return type [dict]

Raises *arango.exceptions.AQLQueryListError* – If retrieval fails.

clear_slow_queries() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear slow AQL queries.

Returns True if slow queries were cleared successfully.

Return type bool

Raises *arango.exceptions.AQLQueryClearError* – If operation fails.

tracking() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return AQL query tracking properties.

Returns AQL query tracking properties.

Return type dict

Raises *arango.exceptions.AQLQueryTrackingGetError* – If retrieval fails.

set_tracking(*enabled: Optional[bool] = None*, *max_slow_queries: Optional[int] = None*, *slow_query_threshold: Optional[int] = None*, *max_query_string_length: Optional[int] = None*, *track_bind_vars: Optional[bool] = None*, *track_slow_queries: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Configure AQL query tracking properties

Parameters

- **enabled** (*bool*) – Track queries if set to True.
- **max_slow_queries** (*int*) – Max number of slow queries to track. Oldest entries are discarded first.
- **slow_query_threshold** (*int*) – Runtime threshold (in seconds) for treating a query as slow.
- **max_query_string_length** (*int*) – Max query string length (in bytes) tracked.
- **track_bind_vars** (*bool*) – If set to True, track bind variables used in queries.
- **track_slow_queries** (*bool*) – If set to True, track slow queries whose runtimes exceed **slow_query_threshold**. To use this, parameter **enabled** must be set to True.

Returns Updated AQL query tracking properties.

Return type dict

Raises *arango.exceptions.AQLQueryTrackingSetError* – If operation fails.

functions() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
List the AQL functions defined in the database.

Returns AQL functions.

Return type [dict]

Raises *arango.exceptions.AQLFunctionListError* – If retrieval fails.

create_function(*name: str*, *code: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new AQL function.

Parameters

- **name** (*str*) – AQL function name.
- **code** (*str*) – Function definition in Javascript.

Returns Whether the AQL function was newly created or an existing one was replaced.

Return type dict

Raises *arango.exceptions.AQLFunctionCreateError* – If create fails.

delete_function(*name: str*, *group: bool = False*, *ignore_missing: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete an AQL function.

Parameters

- **name** (*str*) – AQL function name.
- **group** (*bool*) – If set to True, value of parameter **name** is treated as a namespace prefix, and all functions in the namespace are deleted. If set to False, the value of **name** must be a fully qualified function name including any namespaces.
- **ignore_missing** (*bool*) – Do not raise an exception on missing function.

Returns Number of AQL functions deleted if operation was successful, False if function(s) was not found and **ignore_missing** was set to True.

Return type dict | bool

Raises *arango.exceptions.AQLFunctionDeleteError* – If delete fails.

3.33.5 AQLQueryCache

class arango.aql.**AQLQueryCache**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

AQL Query Cache API wrapper.

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the query cache properties.

Returns Query cache properties.

Return type dict

Raises *arango.exceptions.AQLCachePropertiesError* – If retrieval fails.

configure(*mode: Optional[str] = None*, *max_results: Optional[int] = None*, *max_results_size: Optional[int] = None*, *max_entry_size: Optional[int] = None*, *include_system: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Configure the query cache properties.

Parameters

- **mode** (*str*) – Operation mode. Allowed values are "off", "on" and "demand".
- **max_results** (*int*) – Max number of query results stored per database-specific cache.
- **max_results_size** (*int*) – Max cumulative size of query results stored per database-specific cache.
- **max_entry_size** (*int*) – Max entry size of each query result stored per database-specific cache.
- **include_system** (*bool*) – Store results of queries in system collections.

Returns Query cache properties.

Return type dict

Raises ***arango.exceptions.AQLCacheConfigureError*** – If operation fails.

entries() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the query cache entries.

Returns Query cache entries.

Return type [dict]

Raises ***AQLCacheEntriesError*** – If retrieval fails.

clear() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear the query cache.

Returns True if query cache was cleared successfully.

Return type bool

Raises ***arango.exceptions.AQLCacheClearError*** – If operation fails.

3.33.6 Backup

class arango.backup.**Backup**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

get(*backup_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return backup details.

Parameters **backup_id** (*str*) – If set, details on only the specified backup is returned. Otherwise details on all backups are returned.

Returns Backup details.

Return type dict

Raises ***arango.exceptions.BackupGetError*** – If delete fails.

create(*label: Optional[str] = None*, *allow_inconsistent: Optional[bool] = None*, *force: Optional[bool] = None*, *timeout: Optional[numbers.Number] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a backup when the global write lock can be obtained.

Parameters

- **label** (*str*) – Backup label. If not given, a UUID is used.
- **allow_inconsistent** (*bool*) – Allow inconsistent backup when the global transaction lock cannot be acquired before timeout. Default value is False.
- **force** (*bool*) – Forcefully abort all running transactions to ensure a consistent backup when the global transaction lock cannot be acquired before timeout. Default (and highly recommended) value is False.
- **timeout** (*int*) – Timeout in seconds for creating the backup. Default value is 120 seconds.

Returns Result of the create operation.

Return type dict

Raises ***arango.exceptions.BackupCreateError*** – If create fails.

delete(*backup_id: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a backup.

Parameters **backup_id** (*str*) – Backup ID.

Returns True if the backup was deleted successfully.

Return type bool

Raises ***arango.exceptions.BackupDeleteError*** – If delete fails.

download(*backup_id: Optional[str] = None*, *repository: Optional[str] = None*, *abort: Optional[bool] = None*, *config: Optional[Dict[str, Any]] = None*, *download_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Manage backup downloads.

Parameters

- **backup_id** (*str*) – Backup ID used for scheduling a download. Mutually exclusive with parameter **download_id**.
- **repository** (*str*) – Remote repository URL (e.g. "local://tmp/backups"). Required for scheduling a download and mutually exclusive with parameter **download_id**.
- **abort** (*bool*) – If set to True, running download is aborted. Used with parameter **download_id**.
- **config** (*dict*) – Remote repository configuration. Required for scheduling a download and mutually exclusive with parameter **download_id**.
- **download_id** (*str*) – Download ID. Mutually exclusive with parameters **backup_id**, **repository**, and **config**.

Returns Download details.

Return type dict

Raises ***arango.exceptions.BackupDownloadError*** – If operation fails.

upload(*backup_id: Optional[str] = None*, *repository: Optional[str] = None*, *abort: Optional[bool] = None*, *config: Optional[Dict[str, Any]] = None*, *upload_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Manage backup uploads.

Parameters

- **backup_id** (*str*) – Backup ID used for scheduling an upload. Mutually exclusive with parameter **upload_id**.
- **repository** (*str*) – Remote repository URL (e.g. "local://tmp/backups"). Required for scheduling a upload and mutually exclusive with parameter **upload_id**.
- **config** (*dict*) – Remote repository configuration. Required for scheduling an upload and mutually exclusive with parameter **upload_id**.
- **upload_id** (*str*) – Upload ID. Mutually exclusive with parameters **backup_id**, **repository**, and **config**.
- **abort** (*bool*) – If set to True, running upload is aborted. Used with parameter **upload_id**.

Returns Upload details.

Return type dict

Raises ***arango.exceptions.BackupUploadError*** – If upload operation fails.

restore(*backup_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Restore from a local backup.

Parameters **backup_id** (*str*) – Backup ID.

Returns Result of the restore operation.

Return type dict

Raises ***arango.exceptions.BackupRestoreError*** – If restore fails.

conn

Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context

Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str

db_name

Return the name of the current database.

Returns Database name.

Return type str

username
Return the username.

Returns Username.

Return type str

3.33.7 BatchDatabase

class arango.database.**BatchDatabase**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], return_result: bool*)
Database API wrapper tailored specifically for batch execution.

See *arango.database.StandardDatabase.begin_batch_execution()*.

Parameters

- **connection** – HTTP connection.
- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.BatchJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.

queued_jobs() → Optional[Sequence[arango.job.BatchJob[typing.Any][Any]]]
Return the queued batch jobs.

Returns Queued batch jobs or None if **return_result** parameter was set to False during initialization.

Return type [*arango.job.BatchJob*] | None

commit() → Optional[Sequence[arango.job.BatchJob[typing.Any][Any]]]
Execute the queued requests in a single batch API request.

If **return_result** parameter was set to True during initialization, *arango.job.BatchJob* instances are populated with results.

Returns Batch jobs, or None if **return_result** parameter was set to False during initialization.

Return type [*arango.job.BatchJob*] | None

Raises

- ***arango.exceptions.BatchStateError*** – If batch state is invalid (e.g. batch was already committed or the response size did not match expected).
- ***arango.exceptions.BatchExecuteError*** – If commit fails.

analyzer(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return analyzer details.

Parameters **name** (*str*) – Analyzer name.

Returns Analyzer details.

Return type dict

Raises ***arango.exceptions.AnalyzerGetError*** – If retrieval fails.

analyzers() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of analyzers.

Returns List of analyzers.

Return type [dict]

Raises *arango.exceptions.AnalyzerListError* – If retrieval fails.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status: str*, *count: Optional[int] = None*) → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]
Return IDs of async jobs with given status.

Parameters

- **status** (*str*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str]

Raises *arango.exceptions.AsyncJobListError* – If retrieval fails.

backup
Return Backup API wrapper.

Returns Backup API wrapper.

Return type *arango.backup.Backup*

clear_async_jobs(*threshold: Optional[int] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int | None*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises *arango.exceptions.AsyncJobClearError* – If operation fails.

cluster
Return Cluster API wrapper.

Returns Cluster API wrapper.

Return type *arango.cluster.Cluster*

collection(*name: str*) → arango.collection.StandardCollection
Return the standard collection API wrapper.

Parameters **name** (*str*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

conn
Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str

create_analyzer(*name: str*, *analyzer_type: str*, *properties: Optional[Dict[str, Any]] = None*, *features: Optional[Sequence[str]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **analyzer_type** (*str*) – Analyzer type.
- **properties** (*dict | None*) – Analyzer properties.
- **features** (*list | None*) – Analyzer features.

Returns Analyzer details.

Return type dict

Raises ***arango.exceptions.AnalyzerCreateError*** – If create fails.

create_arangosearch_view(*name: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict | None*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewCreateError*** – If create fails.

create_collection(*name: str*, *sync: bool = False*, *system: bool = False*, *edge: bool = False*, *user_keys: bool = True*, *key_increment: Optional[int] = None*, *key_offset: Optional[int] = None*, *key_generator: str = 'traditional'*, *shard_fields: Optional[Sequence[str]] = None*, *shard_count: Optional[int] = None*, *replication_factor: Optional[int] = None*, *shard_like: Optional[str] = None*, *sync_replication: Optional[bool] = None*, *enforce_replication_factor: Optional[bool] = None*, *sharding_strategy: Optional[str] = None*, *smart_join_attribute: Optional[str] = None*, *write_concern: Optional[int] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[arango.collection.StandardCollection, arango.job.AsyncJob[arango.collection.StandardCollection][arango.collection.StandardCollection], arango.job.BatchJob[arango.collection.StandardCollection][arango.collection.StandardCollection], None]

Create a new collection.

Parameters

- **name** (*str*) – Collection name.
- **sync** (*bool | None*) – If set to True, document operations via the collection will block until synchronized to disk by default.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **key_generator** (*str*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.
- **sharding_strategy** (*str*) – Sharding strategy. Available for ArangoDB version and up only. Possible values are "community-compat", "enterprise-compat", "enterprise-

smart-edge-compat", "hash" and "enterprise-hash-smart-edge". Refer to ArangoDB documentation for more details on each value.

- **smart_join_attribute** (*str*) – Attribute of the collection which must contain the shard key value of the smart join collection. The shard key for the documents must contain the value of this attribute, followed by a colon ":" and the primary key of the document. Requires parameter **shard_like** to be set to the name of another collection, and parameter **shard_fields** to be set to a single shard key attribute, with another colon ":" at the end. Available only for enterprise version of ArangoDB.
- **write_concern** (*int*) – Write concern for the collection. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time. The value of this parameter cannot be larger than that of **replication_factor**. Default value is 1. Used for clusters only.
- **schema** (*dict*) – Optional dict specifying the collection level schema for documents. See ArangoDB documentation for more information on document schema validation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name: str*, *users: Optional[Sequence[Dict[str, Any]]] = None*, *replication_factor: Union[int, str, None] = None*, *write_concern: Optional[int] = None*, *sharding: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Create a new database.

Parameters

- **name** (*str*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.
- **replication_factor** (*int | str*) – Default replication factor for collections created in this database. Special values include "satellite" which replicates the collection to every DBServer, and 1 which disables replication. Used for clusters only.
- **write_concern** (*int*) – Default write concern for collections created in this database. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time, however. Value of this parameter can not be larger than the value of **replication_factor**. Used for clusters only.
- **sharding** (*str*) – Sharding method used for new collections in this database. Allowed values are: "", "flexible" and "single". The first two are equivalent. Used for clusters only.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name: str*, *edge_definitions: Optional[Sequence[Dict[str, Any]]] = None*, *orphan_collections: Optional[Sequence[str]] = None*, *smart: Optional[bool] = None*, *smart_field: Optional[str] = None*, *shard_count: Optional[int] = None*) → Union[arango.graph.Graph, arango.job.AsyncJob[arango.graph.Graph][arango.graph.Graph], arango.job.BatchJob[arango.graph.Graph][arango.graph.Graph], None]

Create a new graph.

Parameters

- **name** (*str*) – Graph name.
- **edge_definitions** (*[dict] | None*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str] | None*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool | None*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | None*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int | None*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name: str*, *command: str*, *params: Optional[Dict[str, Any]] = None*, *period: Optional[int] = None*, *offset: Optional[int] = None*, *task_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new server task.

Parameters

- **name** (*str*) – Name of the server task.

- **command** (*str*) – Javascript command to execute.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **period** (*int | None*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int | None*) – Initial delay before execution in seconds.
- **task_id** (*str | None*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises *arango.exceptions.TaskCreateError* – If create fails.

create_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – Password.
- **active** (*bool | None*) – True if user is active, False otherwise.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name: str*, *view_type: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a view.

Parameters

- **name** (*str*) – View name.
- **view_type** (*str*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return the names all databases.

Returns Database names.

Return type [str]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str

delete_analyzer(*name: str*, *force: bool = False*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **force** (*bool*) – Remove the analyzer configuration even if in use.
- **ignore_missing** (*bool*) – Do not raise an exception on missing analyzer.

Returns True if analyzer was deleted successfully, False if analyzer was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.AnalyzerDeleteError*** – If delete fails.

delete_collection(*name: str*, *ignore_missing: bool = False*, *system: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete the collection.

Parameters

- **name** (*str*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.CollectionDeleteError*** – If delete fails.

delete_database(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete the database.

Parameters

- **name** (*str*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.DatabaseDeleteError*** – If delete fails.

delete_document(*document: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name: str*, *ignore_missing: bool = False*, *drop_collections: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Drop the graph of the given name from the database.

Parameters

- **name** (*str*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool | None*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a server task.

Parameters

- **task_id** (*str*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.TaskDeleteError* – If delete fails.

delete_user(*username: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a user.

Parameters

- **username** (*str*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.UserDeleteError* – If delete fails.

delete_view(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a view.

Parameters

- **name** (*str*) – View name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing view.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document: Dict[str, Any]*, *rev: Optional[str] = None*, *check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return a document.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.

- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

echo() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises ***arango.exceptions.ServerEchoError*** – If retrieval fails.

encryption() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Rotate the user-supplied keys for encryption.

This method is available only for enterprise edition of ArangoDB.

Returns New TLS data.

Return type dict

Raises ***arango.exceptions.ServerEncryptionError*** – If retrieval fails.

engine() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the database engine details.

Returns Database engine details.

Return type dict

Raises ***arango.exceptions.ServerEngineError*** – If retrieval fails.

execute_transaction(*command: str*, *params: Optional[Dict[str, Any]] = None*, *read: Optional[Sequence[str]] = None*, *write: Optional[Sequence[str]] = None*, *sync: Optional[bool] = None*, *timeout: Optional[numbers.Number] = None*, *max_size: Optional[int] = None*, *allow_implicit: Optional[bool] = None*, *intermediate_commit_count: Optional[int] = None*, *intermediate_commit_size: Optional[int] = None*) → Union[Any, arango.job.AsyncJob[typing.Any][Any], arango.job.BatchJob[typing.Any][Any], None]
Execute raw Javascript command in transaction.

Parameters

- **command** (*str*) – Javascript command to execute.
- **read** (*[str] | None*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str] | None*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.

- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **timeout** (*int | None*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int | None*) – Max transaction size limit in bytes.
- **allow_implicit** (*bool | None*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int | None*) – Max number of operations after which an intermediate commit is performed automatically.
- **intermediate_commit_size** (*int | None*) – Max size of operations in bytes after which an intermediate commit is performed automatically.

Returns Return value of **command**.

Return type Any

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name: str*) → arango.graph.Graph
Return the graph API wrapper.

Parameters **name** (*str*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if collection exists in the database.

Parameters **name** (*str*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a database exists.

Parameters **name** (*str*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document: Dict[str, Any], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if a document exists.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if a graph exists in the database.

Parameters **name** (*str*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if user exists.

Parameters **username** (*str*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection: str, document: Dict[str, Any], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False, overwrite_mode: Optional[str] = None, keep_none: Optional[bool] = None, merge: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new document.

Parameters

- **collection** (*str*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.
- **overwrite_mode** (*str | None*) – Overwrite behavior used when the document key exists already. Allowed values are "replace" (replace-insert) or "update" (update-insert). Implicitly sets the value of parameter **overwrite**.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely. Applies only when **overwrite_mode** is set to "update" (update-insert).
- **merge** (*bool | None*) – If set to True (default), sub-dictionaries are merged instead of the new one overwriting the old one. Applies only when **overwrite_mode** is set to "update" (update-insert).

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return information on currently loaded JWT secrets.

Returns Information on currently loaded JWT secrets.

Return type dict

log_levels() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return current logging levels.

Returns Current logging levels.

Return type dict

metrics() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server metrics in Prometheus format.

Returns Server metrics in Prometheus format.

Return type str

name
Return database name.

Returns Database name.

Return type str

permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns Permission for given database or collection.

Return type str

Raises *arango.exceptions.PermissionGetError* – If retrieval fails.

permissions(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return user permissions for all databases and collections.

Parameters **username** (*str*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raises *arango.exceptions.PermissionListError* – If retrieval fails.

pregel

Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return database properties.

Returns Database properties.

Return type dict

Raises *arango.exceptions.DatabasePropertiesError* – If retrieval fails.

read_log(*upto: Union[str, int, None] = None*, *level: Union[str, int, None] = None*, *start: Optional[int] = None*, *size: Optional[int] = None*, *offset: Optional[int] = None*, *search: Optional[str] = None*, *sort: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Read the global log from server.

Parameters

- **upto** (*int | str*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*int | str*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str*) – Return only the log entries containing the given text.

- **sort** (*str*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises *arango.exceptions.ServerReadLogError* – If read fails.

reload_jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Hot-reload JWT secrets.

Calling this without payload reloads JWT secrets from disk. Only files specified via arangod startup option `--server.jwt-secret-keyfile` or `--server.jwt-secret-folder` are used. It is not possible to change the location where files are loaded from without restarting the server.

Returns Information on reloaded JWT secrets.

Return type dict

reload_routing() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises *arango.exceptions.ServerReloadRoutingError* – If reload fails.

reload_tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Reload TLS data (server key, client-auth CA).

Returns New TLS data.

Return type dict

rename_view(*name: str*, *new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Rename a view.

Parameters

- **name** (*str*) – View name.
- **new_name** (*str*) – New view name.

Returns True if view was renamed successfully.

Return type bool

Raises *arango.exceptions.ViewRenameError* – If delete fails.

replace_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace an ArangoSearch view.

Parameters

- **name** (*str*) – View name.

- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replace_document(*document: Dict[str, Any], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username: str, password: str, active: Optional[bool] = None, extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Replace a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replication
Return Replication API wrapper.

Returns Replication API wrapper.

Return type *arango.replication.Replication*

required_db_version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return required version of target database.

Returns Required version of target database.

Return type str

Raises ***arango.exceptions.ServerRequiredDBVersionError*** – If retrieval fails.

reset_permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reset user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server role.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY", "AGENT" (Agency node in a cluster) or "UNDEFINED".

Return type str

Raises ***arango.exceptions.ServerRoleError*** – If retrieval fails.

run_tests(*tests: Sequence[str]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Run available unittests on the server.

Parameters **tests** (*[str]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises ***arango.exceptions.ServerRunTestsError*** – If execution fails.

set_log_levels(***kwargs*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises ***arango.exceptions.ServerShutdownError*** – If shutdown fails.

statistics(*description: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return server statistics.

Returns Server statistics.

Return type dict

Raises ***arango.exceptions.ServerStatisticsError*** – If retrieval fails.

status() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises ***arango.exceptions.ServerStatusError*** – If retrieval fails.

task(*task_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the details of an active server task.

Parameters **task_id** (*str*) – Server task ID.

Returns Server task details.

Return type dict

Raises *arango.exceptions.TaskGetError* – If retrieval fails.

tasks() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises *arango.exceptions.TaskListError* – If retrieval fails.

time() → Union[datetime.datetime, arango.job.AsyncJob[datetime.datetime][datetime.datetime], arango.job.BatchJob[datetime.datetime][datetime.datetime], None]
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return TLS data (server key, client-auth CA).

Returns TLS data.

Return type dict

update_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

update_document(*document: Dict[str, Any], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username: str*, *permission: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Update user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **permission** (*str*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserUpdateError* – If update fails.

update_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user details.

Parameters **username** (*str*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str

users() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return ArangoDB server version.

Returns Server version.

Return type str

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of views and their summaries.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.33.8 BatchJob

class arango.job.**BatchJob**(*response_handler: Callable[[arango.response.Response], T]*)
Job for tracking and retrieving result of batch API execution.

Parameters **response_handler** (*callable*) – HTTP response handler.

id
Return the batch job ID.

Returns Batch job ID.

Return type str

status() → str
Return the batch job status.

Returns Batch job status. Possible values are "pending" (job is still waiting for batch to be committed), or "done" (batch was committed and the job is updated with the result).

Return type str

result() → T
Return the batch job result.

If the job raised an exception, it is propagated up at this point.

Returns Batch job result.

Raises

- *arango.exceptions.ArangoError* – If the job raised an exception.
- *arango.exceptions.BatchJobResultError* – If job result is not available (i.e. batch is not committed yet).

3.33.9 Cluster

class arango.cluster.**Cluster**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

server_id() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return the server ID.

Returns Server ID.

Return type str

Raises *arango.exceptions.ClusterServerIDError* – If retrieval fails.

server_role() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return the server role.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY", "AGENT" (Agency server in a cluster) or "UNDEFINED".

Return type str

Raises *arango.exceptions.ClusterServerRoleError* – If retrieval fails.

server_version(*server_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the version of the given server.

Parameters **server_id** (*str*) – Server ID.

Returns Version of the given server.

Return type dict

Raises *arango.exceptions.ClusterServerVersionError* – If retrieval fails.

server_engine(*server_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the engine details for the given server.

Parameters **server_id** (*str*) – Server ID.

Returns Engine details of the given server.

Return type dict

Raises *arango.exceptions.ClusterServerEngineError* – If retrieval fails.

server_count() → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]
Return the number of servers in the cluster.

Returns Number of servers in the cluster.

Return type int

Raises *arango.exceptions.ClusterServerCountError* – If retrieval fails.

server_statistics(*server_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the statistics for the given server.

Parameters **server_id** (*str*) – Server ID.

Returns Statistics for the given server.

Return type dict

Raises *arango.exceptions.ClusterServerStatisticsError* – If retrieval fails.

health() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the cluster health.

Returns Cluster health.

Return type dict

Raises *arango.exceptions.ClusterHealthError* – If retrieval fails.

toggle_maintenance_mode(*mode: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Enable or disable the cluster supervision (agency) maintenance mode.

Parameters **mode** (*str*) – Maintenance mode. Allowed values are "on" and "off".

Returns Result of the operation.

Return type dict

Raises *arango.exceptions.ClusterMaintenanceModeError* – If toggle fails.

endpoints() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]
Return coordinate endpoints. This method is for clusters only.

Returns List of endpoints.

Return type [str]

Raises **arango.exceptions.ServerEndpointsError** – If retrieval fails.

3.33.10 Collection

class arango.collection.**Collection**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor], name: str*)
Base class for collection API wrappers.

Parameters

- **connection** – HTTP connection.
- **executor** – API executor.
- **name** – Collection name.

name
Return collection name.

Returns Collection name.

Return type str

recalculate_count() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Recalculate the document count.

Returns True if recalculation was successful.

Return type bool

Raises *arango.exceptions.CollectionRecalculateCountError* – If operation fails.

responsible_shard(*document: Dict[str, Any]*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return the ID of the shard responsible for given **document**.

If the document does not exist, return the shard that would be responsible.

Returns Shard ID

Return type str

rename(*new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Rename the collection.

Renames may not be reflected immediately in async execution, batch execution or transactions. It is recommended to initialize new API wrappers after a rename.

Parameters **new_name** (*str*) – New collection name.

Returns True if collection was renamed successfully.

Return type bool

Raises *arango.exceptions.CollectionRenameError* – If rename fails.

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return collection properties.

Returns Collection properties.

Return type dict

Raises *arango.exceptions.CollectionPropertiesError* – If retrieval fails.

configure(*sync: Optional[bool] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Configure collection properties.

Parameters

- **sync** (*bool | None*) – Block until operations are synchronized to disk.
- **schema** (*dict*) – document schema for validation of objects.

Returns New collection properties.

Return type dict

Raises *arango.exceptions.CollectionConfigureError* – If operation fails.

statistics() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return collection statistics.

Returns Collection statistics.

Return type dict

Raises *arango.exceptions.CollectionStatisticsError* – If retrieval fails.

revision() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return collection revision.

Returns Collection revision.

Return type str

Raises ***arango.exceptions.CollectionRevisionError*** – If retrieval fails.

checksum(*with_rev: bool = False*, *with_data: bool = False*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return collection checksum.

Parameters

- **with_rev** (*bool*) – Include document revisions in checksum calculation.
- **with_data** (*bool*) – Include document data in checksum calculation.

Returns Collection checksum.

Return type str

Raises ***arango.exceptions.CollectionChecksumError*** – If retrieval fails.

load() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Load the collection into memory.

Returns True if collection was loaded successfully.

Return type bool

Raises ***arango.exceptions.CollectionLoadError*** – If operation fails.

unload() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Unload the collection from memory.

Returns True if collection was unloaded successfully.

Return type bool

Raises ***arango.exceptions.CollectionUnloadError*** – If operation fails.

truncate() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete all documents in the collection.

Returns True if collection was truncated successfully.

Return type bool

Raises ***arango.exceptions.CollectionTruncateError*** – If operation fails.

count() → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]
Return the total document count.

Returns Total document count.

Return type int

Raises ***arango.exceptions.DocumentCountError*** – If retrieval fails.

has(*document: Union[str, Dict[str, Any]]*, *rev: Optional[str] = None*, *check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a document exists in the collection.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.

- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

ids() → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return the IDs of all documents in the collection.

Returns Document ID cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentIDsError*** – If retrieval fails.

keys() → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return the keys of all documents in the collection.

Returns Document key cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentKeysError*** – If retrieval fails.

all(*skip: Optional[int] = None*, *limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return all documents in the collection.

Parameters

- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentGetError*** – If retrieval fails.

export(*limit: Optional[int] = None*, *count: bool = False*, *batch_size: Optional[int] = None*, *flush: bool = False*, *flush_wait: Optional[int] = None*, *ttl: Optional[numbers.Number] = None*, *filter_fields: Optional[Sequence[str]] = None*, *filter_type: str = 'include'*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Export all documents in the collection using a server cursor.

Parameters

- **flush** (*bool*) – If set to True, flush the write-ahead log prior to the export. If set to False, documents in the write-ahead log during the export are not included in the result.
- **flush_wait** (*int | None*) – Max wait time in seconds for write-ahead log flush.
- **count** (*bool*) – Include the document count in the server cursor.

- **batch_size** (*int | None*) – Max number of documents in the batch fetched by the cursor in one round trip.
- **limit** (*int | None*) – Max number of documents fetched by the cursor.
- **ttl** (*int | float | None*) – Time-to-live for the cursor on the server.
- **filter_fields** (*[str] | None*) – Document fields to filter with.
- **filter_type** (*str*) – Allowed values are "include" or "exclude".

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If export fails.

find(*filters: Dict[str, Any], skip: Optional[int] = None, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return all documents that match the given filters.

Parameters

- **filters** (*dict*) – Document filters.
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_near(*latitude: numbers.Number, longitude: numbers.Number, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents near a given coordinate.

Documents returned are sorted according to distance, with the nearest document being the first. If there are documents of equal distance, they are randomly chosen from the set until the limit is reached. A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_range(*field: str, lower: int, upper: int, skip: Optional[int] = None, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents within a given range in a random order.

A skiplist index must be defined in the collection to use this method.

Parameters

- **field** (*str*) – Document field name.
- **lower** (*int*) – Lower bound (inclusive).
- **upper** (*int*) – Upper bound (exclusive).
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_radius(*latitude: numbers.Number*, *longitude: numbers.Number*, *radius: numbers.Number*, *distance_field: Optional[str] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents within a given radius around a coordinate.

A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.
- **radius** (*int | float*) – Max radius.
- **distance_field** (*str*) – Document field used to indicate the distance to the given coordinate. This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_box(*latitude1: numbers.Number*, *longitude1: numbers.Number*, *latitude2: numbers.Number*, *longitude2: numbers.Number*, *skip: Optional[int] = None*, *limit: Optional[int] = None*, *index: Optional[str] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return all documents in an rectangular area.

Parameters

- **latitude1** (*int | float*) – First latitude.
- **longitude1** (*int | float*) – First longitude.
- **latitude2** (*int | float*) – Second latitude.
- **longitude2** (*int | float*) – Second longitude
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.
- **index** (*str | None*) – ID of the geo index to use (without the collection prefix). This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_by_text(*field: str*, *query: str*, *limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents that match the given fulltext query.

Parameters

- **field** (*str*) – Document field with fulltext index.
- **query** (*str*) – Fulltext query.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

get_many(*documents: Sequence[Union[str, Dict[str, Any]]]*) → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

Return multiple documents ignoring any missing ones.

Parameters **documents** (*[str | dict]*) – List of document keys, IDs or bodies. Document bodies must contain the "_id" or "_key" fields.

Returns Documents. Missing ones are not included.

Return type [dict]

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

random() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return a random document from the collection.

Returns A random document.

Return type dict

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

indexes() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

Return the collection indexes.

Returns Collection indexes.

Return type [dict]

Raises **arango.exceptions.IndexListError** – If retrieval fails.

add_hash_index(*fields: Sequence[str]*, *unique: Optional[bool] = None*, *sparse: Optional[bool] = None*, *deduplicate: Optional[bool] = None*, *name: Optional[str] = None*, *in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new hash index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool | None*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_skiplist_index(*fields: Sequence[str], unique: Optional[bool] = None, sparse: Optional[bool] = None, deduplicate: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new skiplist index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool | None*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_geo_index(*fields: Union[str, Sequence[str]], ordered: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new geo-spatial index.

Parameters

- **fields** (*str | [str]*) – A single document field or a list of document fields. If a single field is given, the field must have values that are lists with at least two floats. Documents with missing fields or invalid values are excluded.
- **ordered** (*bool | None*) – Whether the order is longitude, then latitude.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_fulltext_index(*fields: Sequence[str], min_length: Optional[int] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new fulltext index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **min_length** (*int | None*) – Minimum number of characters to index.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_persistent_index(*fields: Sequence[str], unique: Optional[bool] = None, sparse: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new persistent index.

Unique persistent indexes on non-sharded keys are not supported in a cluster.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – Exclude documents that do not contain at least one of the indexed fields, or documents that have a value of None in any of the indexed fields.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_ttl_index(*fields: Sequence[str], expiry_time: int, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new TTL (time-to-live) index.

Parameters

- **fields** (*[str]*) – Document field to index.

- **expiry_time** (*int*) – Time of expiry in seconds after document creation.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

delete_index(*index_id: str, ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete an index.

Parameters

- **index_id** (*str*) – Index ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing index.

Returns True if index was deleted successfully, False if index was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.IndexDeleteError* – If delete fails.

load_indexes() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Cache all indexes in the collection into memory.

Returns True if index was loaded successfully.

Return type bool

Raises *arango.exceptions.IndexLoadError* – If operation fails.

insert_many(*documents: Sequence[Dict[str, Any]], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]

Insert multiple documents.

Note: If inserting a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were inserted successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single insert operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include bodies of the new documents in the returned metadata. Ignored if parameter **silent** is set to True
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate keys and the existing documents are replaced.
- **return_old** (*bool*) – Include body of the old documents if replaced. Applies only when value of **overwrite** is set to True.

Returns List of document metadata (e.g. document keys, revisions) and any exception, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update_many(*documents: Sequence[Dict[str, Any]], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]

Update multiple documents.

Note: If updating a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were updated successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single update operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – Partial or full documents with the updated values. They must contain the "_id" or "_key" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.

- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises ***arango.exceptions.DocumentUpdateError*** – If update fails.

update_match (*filters: Dict[str, Any], body: Dict[str, Any], limit: Optional[int] = None, keep_none: bool = True, sync: Optional[bool] = None, merge: bool = True*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Update matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single update operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – Full or partial document body with the updates.
- **limit** (*int | None*) – Max number of documents to update. If the limit is lower than the number of matched documents, random documents are chosen. This parameter is not supported on sharded collections.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.

Returns Number of documents updated.

Return type int

Raises ***arango.exceptions.DocumentUpdateError*** – If update fails.

replace_many(*documents: Sequence[Dict[str, Any]], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]

Replace multiple documents.

Note: If replacing a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were replaced successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single replace operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – New documents to replace the old ones with. They must contain the "_id" or "_key" fields. Edge documents must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises ***arango.exceptions.DocumentReplaceError*** – If replace fails.

replace_match(*filters: Dict[str, Any], body: Dict[str, Any], limit: Optional[int] = None, sync: Optional[bool] = None*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Replace matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single replace operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – New document body.
- **limit** (*int | None*) – Max number of documents to replace. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Number of documents replaced.

Return type int

Raises *arango.exceptions.DocumentReplaceError* – If replace fails.

delete_many(*documents: Sequence[Dict[str, Any]], return_old: bool = False, check_rev: bool = True, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]

Delete multiple documents.

Note: If deleting a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were deleted successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single delete operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[str | dict]*) – Document IDs, keys or bodies. Document bodies must contain the "_id" or "_key" fields.
- **return_old** (*bool*) – Include bodies of the old documents in the result.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises *arango.exceptions.DocumentDeleteError* – If delete fails.

delete_match(*filters: Dict[str, Any], limit: Optional[int] = None, sync: Optional[bool] = None*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Delete matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single delete operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **limit** (*int | None*) – Max number of documents to delete. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Number of documents deleted.

Return type int

Raises ***arango.exceptions.DocumentDeleteError*** – If delete fails.

import_bulk(*documents: Sequence[Dict[str, Any]], halt_on_error: bool = True, details: bool = True, from_prefix: Optional[str] = None, to_prefix: Optional[str] = None, overwrite: Optional[bool] = None, on_duplicate: Optional[str] = None, sync: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Insert multiple documents into the collection.

Note: This method is faster than *arango.collection.Collection.insert_many()* but does not return as many details.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single insert operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.
- **halt_on_error** (*bool*) – Halt the entire import on an error.
- **details** (*bool*) – If set to True, the returned result will include an additional list of detailed error messages.
- **from_prefix** (*str*) – String prefix prepended to the value of "_from" field in each edge document inserted. For example, prefix "foo" prepended to "_from": "bar" will result in "_from": "foo/bar". Applies only to edge collections.

- **to_prefix** (*str*) – String prefix prepended to the value of "_to" field in edge document inserted. For example, prefix "foo" prepended to "_to": "bar" will result in "_to": "foo/bar". Applies only to edge collections.
- **overwrite** (*bool*) – If set to True, all existing documents are removed prior to the import. Indexes are still preserved.
- **on_duplicate** (*str*) – Action to take on unique key constraint violations (for documents with "_key" fields). Allowed values are "error" (do not import the new documents and count them as errors), "update" (update the existing documents while preserving any fields missing in the new ones), "replace" (replace the existing documents with new ones), and "ignore" (do not import the new documents and count them as ignored, as opposed to counting them as errors). Options "update" and "replace" may fail on secondary unique key constraint violations.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Result of the bulk import.

Return type dict

Raises ***arango.exceptions.DocumentInsertError*** – If import fails.

3.33.11 Cursor

class arango.cursor.**Cursor**(*connection: arango.connection.BaseConnection, init_data: Dict[str, Any], cursor_type: str = 'cursor'*)

Cursor API wrapper.

Cursors fetch query results from ArangoDB server in batches. Cursor objects are *stateful* as they store the fetched items in-memory. They must not be shared across threads without proper locking mechanism.

Parameters

- **connection** – HTTP connection.
- **init_data** (*dict*) – Cursor initialization data.
- **cursor_type** (*str*) – Cursor type ("cursor" or "export").

id

Return the cursor ID.

Returns Cursor ID.

Return type str

type

Return the cursor type.

Returns Cursor type ("cursor" or "export").

Return type str

batch() → Optional[Deque[Any]]

Return the current batch of results.

Returns Current batch.

Return type collections.deque

has_more() → Optional[bool]

Return True if more results are available on the server.

Returns True if more results are available on the server.

Return type bool

count() → Optional[int]
Return the total number of documents in the entire result set.

Returns Total number of documents, or None if the count option was not enabled during cursor initialization.

Return type int | None

cached() → Optional[bool]
Return True if results are cached.

Returns True if results are cached.

Return type bool

statistics() → Optional[Dict[str, Any]]
Return cursor statistics.

Returns Cursor statistics.

Return type dict

profile() → Optional[Dict[str, Any]]
Return cursor performance profile.

Returns Cursor performance profile.

Return type dict

warnings() → Optional[Sequence[Dict[str, Any]]]
Return any warnings from the query execution.

Returns Warnings, or None if there are none.

Return type [str]

empty() → bool
Check if the current batch is empty.

Returns True if current batch is empty, False otherwise.

Return type bool

next() → Any
Pop the next item from the current batch.

If current batch is empty/depleted, an API request is automatically sent to ArangoDB server to fetch the next batch and update the cursor.

Returns Next item in current batch.

Raises

- **StopIteration** – If the result set is depleted.
- ***arango.exceptions.CursorNextError*** – If batch retrieval fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

pop() → Any
Pop the next item from current batch.

If current batch is empty/depleted, an exception is raised. You must call *arango.cursor.Cursor.fetch()* to manually fetch the next batch from server.

Returns Next item in current batch.

Raises ***arango.exceptions.CursorEmptyError*** – If current batch is empty.

fetch() → Dict[str, Any]
Fetch the next batch from server and update the cursor.

Returns New batch details.

Return type dict

Raises

- ***arango.exceptions.CursorNextError*** – If batch retrieval fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

close(*ignore_missing: bool = False*) → Optional[bool]
Close the cursor and free any server resources tied to it.

Parameters **ignore_missing** (*bool*) – Do not raise exception on missing cursors.

Returns True if cursor was closed successfully, False if cursor was missing on the server and **ignore_missing** was set to True, None if there are no cursors to close server-side (e.g. result set is smaller than the batch size).

Return type bool | None

Raises

- ***arango.exceptions.CursorCloseError*** – If operation fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

3.33.12 DefaultHTTPClient

class arango.http.**DefaultHTTPClient**
Default HTTP client implementation.

create_session(*host: str*) → requests.sessions.Session
Create and return a new session/connection.

Parameters **host** (*str*) – ArangoDB host URL.

Returns requests session object

Return type requests.Session

send_request(*session: requests.sessions.Session*, *method: str*, *url: str*, *headers: Optional[MutableMapping[str, str]] = None*, *params: Optional[MutableMapping[str, str]] = None*, *data: Union[str, requests_toolbelt.multipart.encoder.MultipartEncoder, None] = None*, *auth: Optional[Tuple[str, str]] = None*) → arango.response.Response
Send an HTTP request.

Parameters

- **session** (*requests.Session*) – Requests session object.
- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **url** (*str*) – Request URL.
- **headers** (*dict*) – Request headers.
- **params** (*dict*) – URL (query) parameters.

- **data** (*str | MultipartEncoder | None*) – Request payload.
- **auth** (*tuple*) – Username and password.

Returns HTTP response.

Return type *arango.response.Response*

3.33.13 EdgeCollection

class arango.collection.**EdgeCollection**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor], graph: str, name: str*)

ArangoDB edge collection API wrapper.

Parameters

- **connection** – HTTP connection.
- **executor** – API executor.
- **graph** – Graph name.
- **name** – Edge collection name.

graph

Return the graph name.

Returns Graph name.

Return type str

get(*edge: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return an edge document.

Parameters

- **edge** (*str | dict*) – Edge document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns Edge document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

insert(*edge: Dict[str, Any], sync: Optional[bool] = None, silent: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new edge document.

Parameters

- **edge** (*dict*) – New edge document to insert. It must contain "_from" and "_to" fields. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update(*edge: Dict[str, Any], check_rev: bool = True, keep_none: bool = True, sync: Optional[bool] = None, silent: bool = False, return_old: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Update an edge document.

Parameters

- **edge** (*dict*) – Partial or full edge document with updated values. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.

- **arango.exceptions.DocumentRevisionError** – If revisions mismatch.

replace(*edge: Dict[str, Any], check_rev: bool = True, sync: Optional[bool] = None, silent: bool = False, return_old: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace an edge document.

Parameters

- **edge** (*dict*) – New edge document to replace the old one with. It must contain the "_key" or "_id" field. It must also contain the "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- **arango.exceptions.DocumentReplaceError** – If replace fails.
- **arango.exceptions.DocumentRevisionError** – If revisions mismatch.

delete(*edge: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, sync: Optional[bool] = None, return_old: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete an edge document.

Parameters

- **edge** (*str | dict*) – Edge document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **return_old** (*bool*) – Return body of the old document in the result.

Returns True if edge was deleted successfully, False if edge was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

link (*from_vertex: Union[str, Dict[str, Any]], to_vertex: Union[str, Dict[str, Any]], data: Optional[Dict[str, Any]] = None, sync: Optional[bool] = None, silent: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new edge document linking the given vertices.

Parameters

- **from_vertex** (*str | dict*) – "From" vertex document ID or body with "_id" field.
- **to_vertex** (*str | dict*) – "To" vertex document ID or body with "_id" field.
- **data** (*dict | None*) – Any extra data for the new edge document. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated).
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

edges (*vertex: Union[str, Dict[str, Any]], direction: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the edge documents coming in and/or out of the vertex.

Parameters

- **vertex** (*str | dict*) – Vertex document ID or body with "_id" field.
- **direction** (*str*) – The direction of the edges. Allowed values are "in" and "out". If not set, edges in both directions are returned.

Returns List of edges and statistics.

Return type dict

Raises ***arango.exceptions.EdgeListError*** – If retrieval fails.

3.33.14 Foxx

class arango.foxx.**Foxx**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

Foxx API wrapper.

services(*exclude_system: bool = False*) → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]

List installed services.

Parameters **exclude_system** (*bool*) – If set to True, system services are excluded.

Returns List of installed service.

Return type [dict]

Raises ***arango.exceptions.FoxxServiceListError*** – If retrieval fails.

service(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return service metadata.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceGetError*** – If retrieval fails.

create_service(*mount: str, source: str, config: Optional[Dict[str, Any]] = None, dependencies: Optional[Dict[str, Any]] = None, development: Optional[bool] = None, setup: Optional[bool] = None, legacy: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Install a new service using JSON definition.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.
- **development** (*bool | None*) – Enable development mode.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Install the service in 2.8 legacy compatibility mode.

Returns Service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceCreateError*** – If install fails.

create_service_with_file(*mount: str*, *filename: str*, *development: Optional[bool] = None*, *setup: Optional[bool] = None*, *legacy: Optional[bool] = None*, *config: Optional[Dict[str, Any]] = None*, *dependencies: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Install a new service using a javascript file or zip bundle.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **filename** (*str*) – Full path to the javascript file or zip bundle.
- **development** (*bool | None*) – Enable development mode.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Install the service in 2.8 legacy compatibility mode.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.

Returns Service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceCreateError*** – If install fails.

update_service(*mount: str*, *source: str*, *config: Optional[Dict[str, Any]] = None*, *dependencies: Optional[Dict[str, Any]] = None*, *teardown: Optional[bool] = None*, *setup: Optional[bool] = None*, *legacy: Optional[bool] = None*, *force: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update (upgrade) a service.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.
- **teardown** (*bool | None*) – Run service teardown script.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Update the service in 2.8 legacy compatibility mode.
- **force** (*bool | None*) – Force update if no service is found.

Returns Updated service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceUpdateError*** – If update fails.

update_service_with_file(*mount: str*, *filename: str*, *teardown: Optional[bool] = None*, *setup: Optional[bool] = None*, *legacy: Optional[bool] = None*, *force: Optional[bool] = None*, *config: Optional[Dict[str, Any]] = None*, *dependencies: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update (upgrade) a service using a javascript file or zip bundle.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **filename** (*str*) – Full path to the javascript file or zip bundle.
- **teardown** (*bool | None*) – Run service teardown script.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Update the service in 2.8 legacy compatibility mode.
- **force** (*bool | None*) – Force update if no service is found.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.

Returns Updated service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceUpdateError*** – If update fails.

replace_service(*mount: str*, *source: str*, *config: Optional[Dict[str, Any]] = None*, *dependencies: Optional[Dict[str, Any]] = None*, *teardown: Optional[bool] = None*, *setup: Optional[bool] = None*, *legacy: Optional[bool] = None*, *force: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a service by removing the old one and installing a new one.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.
- **teardown** (*bool | None*) – Run service teardown script.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Replace the service in 2.8 legacy compatibility mode.
- **force** (*bool | None*) – Force install if no service is found.

Returns Replaced service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceReplaceError*** – If replace fails.

replace_service_with_file(*mount: str*, *filename: str*, *teardown: Optional[bool] = None*, *setup: Optional[bool] = None*, *legacy: Optional[bool] = None*, *force: Optional[bool] = None*, *config: Optional[Dict[str, Any]] = None*, *dependencies: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a service using a javascript file or zip bundle.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **filename** (*str*) – Full path to the javascript file or zip bundle.
- **teardown** (*bool | None*) – Run service teardown script.
- **setup** (*bool | None*) – Run service setup script.
- **legacy** (*bool | None*) – Replace the service in 2.8 legacy compatibility mode.
- **force** (*bool | None*) – Force install if no service is found.
- **config** (*dict | None*) – Configuration values.
- **dependencies** (*dict | None*) – Dependency settings.

Returns Replaced service metadata.

Return type dict

Raises ***arango.exceptions.FoxxServiceReplaceError*** – If replace fails.

delete_service(*mount: str*, *teardown: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Uninstall a service.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **teardown** (*bool | None*) – Run service teardown script.

Returns True if service was deleted successfully.

Return type bool

Raises ***arango.exceptions.FoxxServiceDeleteError*** – If delete fails.

config(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return service configuration.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Configuration values.

Return type dict

Raises ***arango.exceptions.FoxxConfigGetError*** – If retrieval fails.

update_config(*mount: str, config: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update service configuration.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **config** (*dict*) – Configuration values. Omitted options are ignored.

Returns Updated configuration values.

Return type dict

Raises *arango.exceptions.FoxxConfigUpdateError* – If update fails.

replace_config(*mount: str, config: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace service configuration.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **config** (*dict*) – Configuration values. Omitted options are reset to their default values or marked as un-configured.

Returns Replaced configuration values.

Return type dict

Raises *arango.exceptions.FoxxConfigReplaceError* – If replace fails.

dependencies(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return service dependencies.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyGetError* – If retrieval fails.

update_dependencies(*mount: str, dependencies: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update service dependencies.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **dependencies** (*dict*) – Dependencies settings. Omitted ones are ignored.

Returns Updated dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyUpdateError* – If update fails.

replace_dependencies(*mount: str, dependencies: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace service dependencies.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **dependencies** (*dict*) – Dependencies settings. Omitted ones are disabled.

Returns Replaced dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyReplaceError* – If replace fails.

enable_development(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Put the service into development mode.

While the service is running in development mode, it is reloaded from the file system, and its setup script (if any) is re-executed every time the service handles a request.

In a cluster with multiple coordinators, changes to the filesystem on one coordinator is not reflected across other coordinators.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxDevModeEnableError* – If operation fails.

disable_development(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Put the service into production mode.

In a cluster with multiple coordinators, the services on all other coordinators are replaced with the version on the calling coordinator.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxDevModeDisableError* – If operation fails.

readme(*mount: str*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]

Return the service readme.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service readme.

Return type str

Raises *arango.exceptions.FoxxReadmeGetError* – If retrieval fails.

swagger(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the Swagger API description for the given service.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Swagger API description.

Return type dict

Raises *arango.exceptions.FoxxSwaggerGetError* – If retrieval fails.

download(*mount: str*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Download service bundle.

When development mode is enabled, a new bundle is created every time. Otherwise, the bundle represents the version of the service installed on the server.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service bundle in raw string form.

Return type str

Raises ***arango.exceptions.FoxxDownloadError*** – If download fails.

commit(*replace: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Commit local service state of the coordinator to the database.

This can be used to resolve service conflicts between coordinators that cannot be fixed automatically due to missing data.

Parameters **replace** (*bool | None*) – Overwrite any existing service files in database.

Returns True if the state was committed successfully.

Return type bool

Raises ***arango.exceptions.FoxxCommitError*** – If commit fails.

scripts(*mount: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
List service scripts.

Parameters **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").

Returns Service scripts.

Return type dict

Raises ***arango.exceptions.FoxxScriptListError*** – If retrieval fails.

run_script(*mount: str, name: str, arg: Any = None*) → Union[Any, arango.job.AsyncJob[typing.Any][Any], arango.job.BatchJob[typing.Any][Any], None]
Run a service script.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **name** (*str*) – Script name.
- **arg** (*Any*) – Arbitrary value passed into the script as first argument.

Returns Result of the script, if any.

Return type Any

Raises ***arango.exceptions.FoxxScriptRunError*** – If script fails.

run_tests(*mount: str, reporter: str = 'default', idiomatic: Optional[bool] = None, output_format: Optional[str] = None, name_filter: Optional[str] = None*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Run service tests.

Parameters

- **mount** (*str*) – Service mount path (e.g "/_admin/aardvark").
- **reporter** (*str*) – Test reporter. Allowed values are "default" (simple list of test cases), "suite" (object of test cases nested in suites), "stream" (raw stream of test results), "xunit" (XUnit or JUnit compatible structure), or "tap" (raw TAP compatible stream).
- **idiomatic** – Use matching format for the reporter, regardless of the value of parameter **output_format**.
- **output_format** (*str*) – Used to further control format. Allowed values are "x-ldjson", "xml" and "text". When using "stream" reporter, setting this to "x-ldjson" returns newline-delimited JSON stream. When using "tap" reporter, setting this to "text" returns plain text TAP report. When using "xunit" reporter, settings this to "xml" returns an XML instead of JSONML.
- **name_filter** (*str*) – Only run tests whose full name (test suite and test case) matches the given string.

Type bool

Returns Reporter output (e.g. raw JSON string, XML, plain text).

Return type str

Raises ***arango.exceptions.FoxxTestRunError*** – If test fails.

3.33.15 Graph

class arango.graph.**Graph**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor], name: str*)

Graph API wrapper.

name

Return the graph name.

Returns Graph name.

Return type str

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return graph properties.

Returns Graph properties.

Return type dict

Raises ***arango.exceptions.GraphPropertiesError*** – If retrieval fails.

has_vertex_collection(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if the graph has the given vertex collection.

Parameters **name** (*str*) – Vertex collection name.

Returns True if vertex collection exists, False otherwise.

Return type bool

vertex_collections() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return vertex collections in the graph that are not orphaned.

Returns Names of vertex collections that are not orphaned.

Return type [str]

Raises *arango.exceptions.VertexCollectionListError* – If retrieval fails.

vertex_collection(*name: str*) → arango.collection.VertexCollection
Return the vertex collection API wrapper.

Parameters **name** (*str*) – Vertex collection name.

Returns Vertex collection API wrapper.

Return type *arango.collection.VertexCollection*

create_vertex_collection(*name: str*) → Union[arango.collection.VertexCollection, arango.job.AsyncJob[arango.collection.VertexCollection][arango.collection.VertexCollectio arango.job.BatchJob[arango.collection.VertexCollection][arango.collection.VertexCollectio None]
Create a vertex collection in the graph.

Parameters **name** (*str*) – Vertex collection name.

Returns Vertex collection API wrapper.

Return type *arango.collection.VertexCollection*

Raises *arango.exceptions.VertexCollectionCreateError* – If create fails.

delete_vertex_collection(*name: str*, *purge: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Remove a vertex collection from the graph.

Parameters

- **name** (*str*) – Vertex collection name.
- **purge** (*bool*) – If set to True, the vertex collection is not just deleted from the graph but also from the database completely.

Returns True if vertex collection was deleted successfully.

Return type bool

Raises *arango.exceptions.VertexCollectionDeleteError* – If delete fails.

has_edge_definition(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if the graph has the given edge definition.

Parameters **name** (*str*) – Edge collection name.

Returns True if edge definition exists, False otherwise.

Return type bool

has_edge_collection(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if the graph has the given edge collection.

Parameters **name** (*str*) – Edge collection name.

Returns True if edge collection exists, False otherwise.

Return type bool

edge_collection (*name: str*) → arango.collection.EdgeCollection
Return the edge collection API wrapper.

Parameters **name** (*str*) – Edge collection name.

Returns Edge collection API wrapper.

Return type *arango.collection.EdgeCollection*

edge_definitions () → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the edge definitions of the graph.

Returns Edge definitions of the graph.

Return type [dict]

Raises ***arango.exceptions.EdgeDefinitionListError*** – If retrieval fails.

create_edge_definition (*edge_collection: str, from_vertex_collections: Sequence[str], to_vertex_collections: Sequence[str]*) → Union[arango.collection.EdgeCollection, arango.job.AsyncJob[arango.collection.EdgeCollection][arango.collection.EdgeCollection], arango.job.BatchJob[arango.collection.EdgeCollection][arango.collection.EdgeCollection], None]
Create a new edge definition.

An edge definition consists of an edge collection, "from" vertex collection(s) and "to" vertex collection(s). Here is an example entry:

```
{
    'edge_collection': 'edge_collection_name',
    'from_vertex_collections': ['from_vertex_collection_name'],
    'to_vertex_collections': ['to_vertex_collection_name']
}
```

Parameters

- **edge_collection** (*str*) – Edge collection name.
- **from_vertex_collections** (*[str]*) – Names of "from" vertex collections.
- **to_vertex_collections** (*[str]*) – Names of "to" vertex collections.

Returns Edge collection API wrapper.

Return type *arango.collection.EdgeCollection*

Raises ***arango.exceptions.EdgeDefinitionCreateError*** – If create fails.

replace_edge_definition (*edge_collection: str, from_vertex_collections: Sequence[str], to_vertex_collections: Sequence[str]*) → Union[arango.collection.EdgeCollection, arango.job.AsyncJob[arango.collection.EdgeCollection][arango.collection.EdgeCollection], arango.job.BatchJob[arango.collection.EdgeCollection][arango.collection.EdgeCollection], None]
Replace an edge definition.

Parameters

- **edge_collection** (*str*) – Edge collection name.
- **from_vertex_collections** (*[str]*) – Names of "from" vertex collections.

- **to_vertex_collections** (*[str]*) – Names of "to" vertex collections.

Returns Edge collection API wrapper.

Return type *arango.collection.EdgeCollection*

Raises **arango.exceptions.EdgeDefinitionReplaceError** – If replace fails.

delete_edge_definition(*name: str, purge: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete an edge definition from the graph.

Parameters

- **name** (*str*) – Edge collection name.
- **purge** (*bool*) – If set to True, the edge definition is not just removed from the graph but the edge collection is also deleted completely from the database.

Returns True if edge definition was deleted successfully.

Return type bool

Raises **arango.exceptions.EdgeDefinitionDeleteError** – If delete fails.

traverse(*start_vertex: Union[str, Dict[str, Any]], direction: str = 'outbound', item_order: str = 'forward', strategy: Optional[str] = None, order: Optional[str] = None, edge_uniqueness: Optional[str] = None, vertex_uniqueness: Optional[str] = None, max_iter: Optional[int] = None, min_depth: Optional[int] = None, max_depth: Optional[int] = None, init_func: Optional[str] = None, sort_func: Optional[str] = None, filter_func: Optional[str] = None, visitor_func: Optional[str] = None, expander_func: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Traverse the graph and return the visited vertices and edges.

Parameters

- **start_vertex** (*str | dict*) – Start vertex document ID or body with "_id" field.
- **direction** (*str*) – Traversal direction. Allowed values are "outbound" (default), "inbound" and "any".
- **item_order** (*str*) – Item iteration order. Allowed values are "forward" (default) and "backward".
- **strategy** (*str | None*) – Traversal strategy. Allowed values are "depthfirst" and "breadthfirst".
- **order** (*str | None*) – Traversal order. Allowed values are "preorder", "postorder", and "preorder-expander".
- **edge_uniqueness** (*str | None*) – Uniqueness for visited edges. Allowed values are "global", "path" or "none".
- **vertex_uniqueness** (*str | None*) – Uniqueness for visited vertices. Allowed values are "global", "path" or "none".
- **max_iter** (*int | None*) – If set, halt the traversal after the given number of iterations. This parameter can be used to prevent endless loops in cyclic graphs.
- **min_depth** (*int | None*) – Minimum depth of the nodes to visit.
- **max_depth** (*int | None*) – Maximum depth of the nodes to visit.

- **init_func** (*str | None*) – Initialization function in Javascript with signature `(config, result) -> void`. This function is used to initialize values in the result.
- **sort_func** (*str | None*) – Sorting function in Javascript with signature `(left, right) -> integer`, which returns `-1` if `left < right`, `+1` if `left > right` and `0` if `left == right`.
- **filter_func** (*str | None*) – Filter function in Javascript with signature `(config, vertex, path) -> mixed`, where `mixed` can have one of the following values (or an array with multiple): "exclude" (do not visit the vertex), "prune" (do not follow the edges of the vertex), or "undefined" (visit the vertex and follow its edges).
- **visitor_func** (*str | None*) – Visitor function in Javascript with signature `(config, result, vertex, path, connected) -> void`. The return value is ignored, `result` is modified by reference, and `connected` is populated only when parameter **order** is set to "preorder-expander".
- **expander_func** (*str | None*) – Expander function in Javascript with signature `(config, vertex, path) -> mixed`. The function must return an array of connections for `vertex`. Each connection is an object with attributes "edge" and "vertex".

Returns Visited edges and vertices.

Return type dict

Raises ***arango.exceptions.GraphTraverseError*** – If traversal fails.

has_vertex(*vertex: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if the given vertex document exists in the graph.

Parameters

- **vertex** (*str | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns True if vertex document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentGetError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

vertex(*vertex: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return a vertex document.

Parameters

- **vertex** (*str | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.

- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns Vertex document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

insert_vertex(*collection: str, vertex: Dict[str, Any], sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new vertex document.

Parameters

- **collection** (*str*) – Vertex collection name.
- **vertex** (*dict*) – New vertex document to insert. If it has "_key" or "_id" field, its value is used as key of the new vertex (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update_vertex(*vertex: Dict[str, Any], check_rev: bool = True, keep_none: bool = True, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Update a vertex document.

Parameters

- **vertex** (*dict*) – Partial or full vertex document with updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_vertex(*vertex: Dict[str, Any], check_rev: bool = True, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace a vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to replace the old one with. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_vertex(*vertex: Dict[str, Any], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, sync: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a vertex document.

Parameters

- **vertex** (*str | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns True if vertex was deleted successfully, False if vertex was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_edge(*edge: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if the given edge document exists in the graph.

Parameters

- **edge** (*str | dict*) – Edge document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns True if edge document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

edge(*edge: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return an edge document.

Parameters

- **edge** (*str | dict*) – Edge document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns Edge document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

insert_edge(*collection: str, edge: Dict[str, Any], sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new edge document.

Parameters

- **collection** (*str*) – Edge collection name.

- **edge** (*dict*) – New edge document to insert. It must contain "_from" and "_to" fields. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update_edge(*edge: Dict[str, Any], check_rev: bool = True, keep_none: bool = True, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Update an edge document.

Parameters

- **edge** (*dict*) – Partial or full edge document with updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_edge(*edge: Dict[str, Any], check_rev: bool = True, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace an edge document.

Parameters

- **edge** (*dict*) – New edge document to replace the old one with. It must contain the "_id" field. It must also contain the "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_edge(*edge: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, sync: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete an edge document.

Parameters

- **edge** (*str | dict*) – Edge document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns True if edge was deleted successfully, False if edge was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

link(*collection: str, from_vertex: Union[str, Dict[str, Any]], to_vertex: Union[str, Dict[str, Any]], data: Optional[Dict[str, Any]] = None, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new edge document linking the given vertices.

Parameters

- **collection** (*str*) – Edge collection name.
- **from_vertex** (*str | dict*) – "From" vertex document ID or body with "_id" field.
- **to_vertex** (*str | dict*) – "To" vertex document ID or body with "_id" field.
- **data** (*dict*) – Any extra data for the new edge document. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated).

- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

edges (*collection: str, vertex: Union[str, Dict[str, Any]], direction: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the edge documents coming in and/or out of given vertex.

Parameters

- **collection** (*str*) – Edge collection name.
- **vertex** (*str | dict*) – Vertex document ID or body with "_id" field.
- **direction** (*str*) – The direction of the edges. Allowed values are "in" and "out". If not set, edges in both directions are returned.

Returns List of edges and statistics.

Return type dict

Raises ***arango.exceptions.EdgeListError*** – If retrieval fails.

3.33.16 HTTPClient

class arango.http.**HTTPClient**

Abstract base class for HTTP clients.

create_session (*host: str*) → requests.sessions.Session

Return a new requests session given the host URL.

This method must be overridden by the user.

Parameters **host** (*str*) – ArangoDB host URL.

Returns Requests session object.

Return type requests.Session

send_request (*session: requests.sessions.Session, method: str, url: str, headers: Optional[MutableMapping[str, str]] = None, params: Optional[MutableMapping[str, str]] = None, data: Union[str, requests_toolbelt.multipart.encoder.MultipartEncoder, None] = None, auth: Optional[Tuple[str, str]] = None*) → arango.response.Response

Send an HTTP request.

This method must be overridden by the user.

Parameters

- **session** (*requests.Session*) – Requests session object.
- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **url** (*str*) – Request URL.
- **headers** (*dict*) – Request headers.

- **params** (*dict*) – URL (query) parameters.
- **data** (*str | MultipartEncoder | None*) – Request payload.
- **auth** (*tuple*) – Username and password.

Returns HTTP response.

Return type *arango.response.Response*

3.33.17 Pregel

class arango.pregel.**Pregel**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

Pregel API wrapper.

job(*job_id: int*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the details of a Pregel job.

Parameters **job_id** (*int*) – Pregel job ID.

Returns Details of the Pregel job.

Return type dict

Raises ***arango.exceptions.PregelJobGetError*** – If retrieval fails.

create_job(*graph: str, algorithm: str, store: bool = True, max_gss: Optional[int] = None, thread_count: Optional[int] = None, async_mode: Optional[bool] = None, result_field: Optional[str] = None, algorithm_params: Optional[Dict[str, Any]] = None*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Start a new Pregel job.

Parameters

- **graph** (*str*) – Graph name.
- **algorithm** (*str*) – Algorithm (e.g. "pagerank").
- **store** (*bool*) – If set to True, Pregel engine writes results back to the database. If set to False, results can be queried via AQL.
- **max_gss** (*int | None*) – Max number of global iterations for the algorithm.
- **thread_count** (*int | None*) – Number of parallel threads to use per worker. This does not influence the number of threads used to load or store data from the database (it depends on the number of shards).
- **async_mode** (*bool | None*) – If set to True, algorithms which support async mode run without synchronized global iterations. This might lead to performance increase if there are load imbalances.
- **result_field** (*str | None*) – If specified, most algorithms will write their results into this field.
- **algorithm_params** (*dict | None*) – Additional algorithm parameters.

Returns Pregel job ID.

Return type int

Raises ***arango.exceptions.PregelJobCreateError*** – If create fails.

delete_job(*job_id: int*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete a Pregel job.

Parameters **job_id** (*int*) – Pregel job ID.

Returns True if Pregel job was deleted successfully.

Return type bool

Raises ***arango.exceptions.PregelJobDeleteError*** – If delete fails.

3.33.18 Replication

class arango.replication.**Replication**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

inventory(*batch_id: str*, *include_system: Optional[bool] = None*, *all_databases: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return an overview of collections and indexes.

Parameters

- **batch_id** (*str*) – Batch ID.
- **include_system** (*bool | None*) – Include system collections in the result. Default value is True.
- **all_databases** (*bool | None*) – Include all databases. Only works on "_system" database. Default value is False.

Returns Overview of collections and indexes.

Return type dict

Raises ***arango.exceptions.ReplicationInventoryError*** – If retrieval fails.

create_dump_batch(*ttl: Optional[int] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create a new dump batch.

Parameters **ttl** (*int | None*) – Time-to-live for the new batch in seconds.

Returns ID of the batch.

Return type dict

Raises ***arango.exceptions.ReplicationDumpBatchCreateError*** – If create fails.

delete_dump_batch(*batch_id: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete a dump batch.

Parameters **batch_id** (*str*) – Dump batch ID.

Returns True if deletion was successful.

Return type bool

Raises *arango.exceptions.ReplicationDumpBatchDeleteError* – If delete fails.

extend_dump_batch(*batch_id: str*, *ttl: int*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Extend a dump batch.

Parameters

- **batch_id** (*str*) – Dump batch ID.
- **ttl** (*int*) – Time-to-live for the new batch in seconds.

Returns True if operation was successful.

Return type bool

Raises *arango.exceptions.ReplicationDumpBatchExtendError* – If dump fails.

dump(*collection: str*, *batch_id: Optional[str] = None*, *chunk_size: Optional[int] = None*, *deserialize: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the events data of one collection.

Parameters

- **collection** (*str*) – Name or ID of the collection to dump.
- **batch_id** (*str | None*) – Batch ID.
- **chunk_size** (*int | None*) – Size of the result in bytes. This value is honored approximately only.
- **deserialize** (*bool*) – Deserialize the response content. Default is False.

Returns Collection events data.

Return type str | [dict]

Raises *arango.exceptions.ReplicationDumpError* – If retrieval fails.

synchronize(*endpoint: str*, *database: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *include_system: Optional[bool] = None*, *incremental: Optional[bool] = None*, *restrict_type: Optional[str] = None*, *restrict_collections: Optional[Sequence[str]] = None*, *initial_sync_wait_time: Optional[int] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Synchronize data from a remote endpoint.

Parameters

- **endpoint** (*str*) – Master endpoint (e.g. "tcp://192.168.173.13:8529").
- **database** (*str | None*) – Database name.
- **username** (*str | None*) – Username.
- **password** (*str | None*) – Password.
- **include_system** (*bool | None*) – Whether to include system collection operations.

- **incremental** (*bool | None*) – If set to True, then an incremental synchronization method is used for synchronizing data in collections. This method is useful when collections already exist locally, and only the remaining differences need to be transferred from the remote endpoint. In this case, the incremental synchronization can be faster than a full synchronization. Default value is False, meaning complete data is transferred.
- **restrict_type** (*str | None*) – Optional string value for collection filtering. Allowed values are "include" or "exclude".
- **restrict_collections** (*[str] | None*) – Optional list of collections for use with argument **restrict_type**. If **restrict_type** set to "include", only the specified collections are synchronised. Otherwise, all but the specified ones are synchronized.
- **initial_sync_wait_time** (*int | None*) – Maximum wait time in seconds that the initial synchronization will wait for a response from master when fetching collection data. This can be used to control after what time the initial synchronization will give up waiting for response and fail. Value is ignored if set to 0.

Returns Collections transferred and last log tick.

Return type dict

Raises ***arango.exceptions.ReplicationSyncError*** – If sync fails.

cluster_inventory(*include_system: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return an overview of collections and indexes in a cluster.

Parameters **include_system** (*bool*) – Include system collections in the result. Default value is True.

Returns Overview of collections and indexes on the cluster.

Return type dict

Raises ***arango.exceptions.ReplicationClusterInventoryError*** – If retrieval fails.

logger_state() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the state of the replication logger.

Returns Logger state.

Return type dict

Raises ***arango.exceptions.ReplicationLoggerStateError*** – If retrieval fails.

logger_first_tick() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]

Return the first available tick value from the server.

Returns First tick value.

Return type str

Raises ***arango.exceptions.ReplicationLoggerFirstTickError*** – If retrieval fails.

applier_config() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the configuration of the replication applier.

Returns Configuration of the replication applier.

Return type dict

Raises *arango.exceptions.ReplicationApplierConfigError* – If retrieval fails.

set_applier_config(*endpoint: str*, *database: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *max_connect_retries: Optional[int] = None*, *connect_timeout: Optional[int] = None*, *request_timeout: Optional[int] = None*, *chunk_size: Optional[int] = None*, *auto_start: Optional[bool] = None*, *adaptive_polling: Optional[bool] = None*, *include_system: Optional[bool] = None*, *auto_resync: Optional[bool] = None*, *auto_resync_retries: Optional[int] = None*, *initial_sync_max_wait_time: Optional[int] = None*, *connection_retry_wait_time: Optional[int] = None*, *idle_min_wait_time: Optional[int] = None*, *idle_max_wait_time: Optional[int] = None*, *require_from_present: Optional[bool] = None*, *verbose: Optional[bool] = None*, *restrict_type: Optional[str] = None*, *restrict_collections: Optional[Sequence[str]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Set configuration values of the replication applier.

Parameters

- **endpoint** (*str*) – Server endpoint (e.g. "tcp://192.168.173.13:8529").
- **database** (*str | None*) – Database name.
- **username** (*str | None*) – Username.
- **password** (*str | None*) – Password.
- **max_connect_retries** (*int | None*) – Maximum number of connection attempts the applier makes in a row before stopping itself.
- **connect_timeout** (*int | None*) – Timeout in seconds when attempting to connect to the endpoint. This value is used for each connection attempt.
- **request_timeout** (*int | None*) – Timeout in seconds for individual requests to the endpoint.
- **chunk_size** (*int | None*) – Requested maximum size in bytes for log transfer packets when the endpoint is contacted.
- **auto_start** (*bool | None*) – Whether to auto-start the replication applier on (next and following) server starts.
- **adaptive_polling** (*bool | None*) – If set to True, replication applier sleeps for an increasingly long period in case the logger server at the endpoint has no replication events to apply. Using adaptive polling reduces the amount of work done by both the applier and the logger server when there are infrequent changes. The downside is that it might take longer for the replication applier to detect new events.
- **include_system** (*bool | None*) – Whether system collection operations are applied.
- **auto_resync** (*bool | None*) – Whether the slave should perform a full automatic resynchronization with the master in case the master cannot serve log data requested by the slave, or when the replication is started and no tick value can be found.
- **auto_resync_retries** (*int | None*) – Max number of resynchronization retries. Setting this to 0 disables it.

- **initial_sync_max_wait_time** (*int | None*) – Max wait time in seconds the initial synchronization waits for master on collection data. This value is relevant even for continuous replication when **auto_resync** is set to True because this may re-start the initial synchronization when master cannot provide log data slave requires. This value is ignored if set to 0.
- **connection_retry_wait_time** (*int | None*) – Time in seconds the applier idles before trying to connect to master in case of connection problems. This value is ignored if set to 0.
- **idle_min_wait_time** (*int | None*) – Minimum wait time in seconds the applier idles before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. This value is ignored if set to 0.
- **idle_max_wait_time** (*int | None*) – Maximum wait time in seconds the applier idles before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the maximum frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. Applies only when argument **adaptive_polling** is set to True. This value is ignored if set to 0.
- **require_from_present** (*bool | None*) – If set to True, replication applier checks at start whether the start tick from which it starts or resumes replication is still present on the master. If not, then there would be data loss. If set to True, the replication applier aborts with an appropriate error message. If set to False, the applier still starts and ignores the data loss.
- **verbose** (*bool | None*) – If set to True, a log line is emitted for all operations performed by the replication applier. This should be used for debugging replication problems only.
- **restrict_type** (*str | None*) – Optional string value for collection filtering. Allowed values are "include" or "exclude".
- **restrict_collections** (*[str] | None*) – Optional list of collections for use with argument **restrict_type**. If **restrict_type** set to "include", only the specified collections are included. Otherwise, only the specified collections are excluded.

Returns Updated configuration.

Return type dict

Raises ***arango.exceptions.ReplicationApplierConfigSetError*** – If update fails.

applier_state() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return the state of the replication applier

Returns Applier state and details.

Return type dict

Raises ***arango.exceptions.ReplicationApplierStateError*** – If retrieval fails.

start_applier(*last_tick: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Start the replication applier.

Parameters **last_tick** (*str*) – The remote last log tick value from which to start applying replication. If not specified, the last saved tick from the previous applier run is used. If there is no previous applier state saved, the applier starts at the beginning of the logger server's log.

Returns Applier state and details.

Return type dict

Raises *arango.exceptions.ReplicationApplierStartError* – If operation fails.

stop_applier() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Stop the replication applier.

Returns Applier state and details.

Return type dict

Raises *arango.exceptions.ReplicationApplierStopError* – If operation fails.

make_slave(*endpoint: str*, *database: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *restrict_type: Optional[str] = None*, *restrict_collections: Optional[Sequence[str]] = None*, *include_system: Optional[bool] = None*, *max_connect_retries: Optional[int] = None*, *connect_timeout: Optional[int] = None*, *request_timeout: Optional[int] = None*, *chunk_size: Optional[int] = None*, *adaptive_polling: Optional[bool] = None*, *auto_resync: Optional[bool] = None*, *auto_resync_retries: Optional[int] = None*, *initial_sync_max_wait_time: Optional[int] = None*, *connection_retry_wait_time: Optional[int] = None*, *idle_min_wait_time: Optional[int] = None*, *idle_max_wait_time: Optional[int] = None*, *require_from_present: Optional[bool] = None*, *verbose: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Change the server role to slave.

Parameters

- **endpoint** (*str*) – Master endpoint (e.g. "tcp://192.168.173.13:8529").
- **database** (*str | None*) – Database name.
- **username** (*str | None*) – Username.
- **password** (*str | None*) – Password.
- **restrict_type** (*str | None*) – Optional string value for collection filtering. Allowed values are "include" or "exclude".
- **restrict_collections** (*[str] | None*) – Optional list of collections for use with argument **restrict_type**. If **restrict_type** set to "include", only the specified collections are included. Otherwise, only the specified collections are excluded.
- **include_system** (*bool | None*) – Whether system collection operations are applied.
- **max_connect_retries** (*int | None*) – Maximum number of connection attempts the applier makes in a row before stopping itself.
- **connect_timeout** (*int | None*) – Timeout in seconds when attempting to connect to the endpoint. This value is used for each connection attempt.

- **request_timeout** (*int | None*) – Timeout in seconds for individual requests to the endpoint.
- **chunk_size** (*int | None*) – Requested maximum size in bytes for log transfer packets when the endpoint is contacted.
- **adaptive_polling** (*bool | None*) – If set to True, replication applier sleeps for an increasingly long period in case the logger server at the endpoint has no replication events to apply. Using adaptive polling reduces the amount of work done by both the applier and the logger server when there are infrequent changes. The downside is that it might take longer for the replication applier to detect new events.
- **auto_resync** (*bool | None*) – Whether the slave should perform a full automatic resynchronization with the master in case the master cannot serve log data requested by the slave, or when the replication is started and no tick value can be found.
- **auto_resync_retries** (*int | None*) – Max number of resynchronization retries. Setting this to 0 disables it.
- **initial_sync_max_wait_time** (*int | None*) – Max wait time in seconds the initial synchronization waits for master on collection data. This value is relevant even for continuous replication when **auto_resync** is set to True because this may restart the initial synchronization when master cannot provide log data slave requires. This value is ignored if set to 0.
- **connection_retry_wait_time** (*int | None*) – Time in seconds the applier idles before trying to connect to master in case of connection problems. This value is ignored if set to 0.
- **idle_min_wait_time** (*int | None*) – Minimum wait time in seconds the applier idles before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. This value is ignored if set to 0.
- **idle_max_wait_time** (*int | None*) – Maximum wait time in seconds the applier idles before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the maximum frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. Applies only when argument **adaptive_polling** is set to True. This value is ignored if set to 0.
- **require_from_present** (*bool | None*) – If set to True, replication applier checks at start whether the start tick from which it starts or resumes replication is still present on the master. If not, then there would be data loss. If set to True, the replication applier aborts with an appropriate error message. If set to False, the applier still starts and ignores the data loss.
- **verbose** (*bool | None*) – If set to True, a log line is emitted for all operations performed by the replication applier. This should be used for debugging replication problems only.

Returns Replication details.

Return type dict

Raises *arango.exceptions.ReplicationApplierStopError* – If operation fails.

server_id() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return this server's ID.

Returns Server ID.

Return type str

Raises *arango.exceptions.ReplicationServerIDError* – If retrieval fails.

3.33.19 Request

class arango.request.**Request**(*method: str, endpoint: str, headers: Optional[MutableMapping[str, str]] = None, params: Optional[MutableMapping[str, Union[bool, int, str]]] = None, data: Any = None, read: Union[str, Sequence[str], None] = None, write: Union[str, Sequence[str], None] = None, exclusive: Union[str, Sequence[str], None] = None, deserialize: bool = True*)

HTTP request.

Parameters

- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **endpoint** (*str*) – API endpoint.
- **headers** (*dict | None*) – Request headers.
- **params** (*dict | None*) – URL parameters.
- **data** (*str | bool | int | float | list | dict | None | MultipartEncoder*) – Request payload.
- **read** (*str | [str] | None*) – Names of collections read during transaction.
- **write** (*str | [str] | None*) – Name(s) of collections written to during transaction with shared access.
- **exclusive** (*str | [str] | None*) – Name(s) of collections written to during transaction with exclusive access.
- **deserialize** (*bool*) – Whether the response body can be deserialized.

Variables

- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **endpoint** (*str*) – API endpoint.
- **headers** (*dict | None*) – Request headers.
- **params** (*dict | None*) – URL (query) parameters.
- **data** (*str | bool | int | float | list | dict | None*) – Request payload.
- **read** (*str | [str] | None*) – Names of collections read during transaction.
- **write** (*str | [str] | None*) – Name(s) of collections written to during transaction with shared access.
- **exclusive** (*str | [str] | None*) – Name(s) of collections written to during transaction with exclusive access.
- **deserialize** (*bool*) – Whether the response body can be deserialized.

3.33.20 Response

class arango.response.**Response**(*method: str, url: str, headers: MutableMapping[str, str], status_code: int, status_text: str, raw_body: str*)

HTTP response.

Parameters

- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **url** (*str*) – API URL.
- **headers** (*MutableMapping*) – Response headers.
- **status_code** (*int*) – Response status code.
- **status_text** (*str*) – Response status text.
- **raw_body** (*str*) – Raw response body.

Variables

- **method** (*str*) – HTTP method in lowercase (e.g. "post").
- **url** (*str*) – API URL.
- **headers** (*MutableMapping*) – Response headers.
- **status_code** (*int*) – Response status code.
- **status_text** (*str*) – Response status text.
- **raw_body** (*str*) – Raw response body.
- **body** (*str | bool | int | float | list | dict | None*) – JSON-deserialized response body.
- **error_code** (*int*) – Error code from ArangoDB server.
- **error_message** (*str*) – Error message from ArangoDB server.
- **is_success** (*bool*) – True if response status code was 2XX.

3.33.21 StandardCollection

class arango.collection.**StandardCollection**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor], name: str*)

Standard ArangoDB collection API wrapper.

get(*document: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.

- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

insert(*document: Dict[str, Any], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False, overwrite_mode: Optional[str] = None, keep_none: Optional[bool] = None, merge: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new document.

Parameters

- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and existing document is overwritten (replace-insert).
- **return_old** (*bool*) – Include body of the old document if overwritten. Ignored if parameter **silent** is set to True.
- **overwrite_mode** (*str | None*) – Overwrite behavior used when the document key exists already. Allowed values are "replace" (replace-insert) or "update" (update-insert). Implicitly sets the value of parameter **overwrite**.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely. Applies only when **overwrite_mode** is set to "update" (update-insert).
- **merge** (*bool | None*) – If set to True (default), sub-dictionaries are merged instead of the new one overwriting the old one. Applies only when **overwrite_mode** is set to "update" (update-insert).

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update(*document: Dict[str, Any], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" or "_key" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace(*document: Dict[str, Any], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" or "_key" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.

- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete(*document: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

add_fulltext_index(*fields: Sequence[str], min_length: Optional[int] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new fulltext index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **min_length** (*int | None*) – Minimum number of characters to index.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_geo_index(*fields: Union[str, Sequence[str]], ordered: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new geo-spatial index.

Parameters

- **fields** (*str | [str]*) – A single document field or a list of document fields. If a single field is given, the field must have values that are lists with at least two floats. Documents with missing fields or invalid values are excluded.
- **ordered** (*bool | None*) – Whether the order is longitude, then latitude.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_hash_index(*fields: Sequence[str], unique: Optional[bool] = None, sparse: Optional[bool] = None, deduplicate: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new hash index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool | None*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_persistent_index(*fields: Sequence[str], unique: Optional[bool] = None, sparse: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new persistent index.

Unique persistent indexes on non-sharded keys are not supported in a cluster.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – Exclude documents that do not contain at least one of the indexed fields, or documents that have a value of None in any of the indexed fields.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises ***arango.exceptions.IndexCreateError*** – If create fails.

add_skiplist_index(*fields: Sequence[str], unique: Optional[bool] = None, sparse: Optional[bool] = None, deduplicate: Optional[bool] = None, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new skiplist index.

Parameters

- **fields** (*[str]*) – Document fields to index.
- **unique** (*bool | None*) – Whether the index is unique.
- **sparse** (*bool | None*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool | None*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises ***arango.exceptions.IndexCreateError*** – If create fails.

add_ttl_index(*fields: Sequence[str], expiry_time: int, name: Optional[str] = None, in_background: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new TTL (time-to-live) index.

Parameters

- **fields** (*[str]*) – Document field to index.
- **expiry_time** (*int*) – Time of expiry in seconds after document creation.
- **name** (*str | None*) – Optional name for the index.
- **in_background** (*bool | None*) – Do not hold the collection lock.

Returns New index details.

Return type dict

Raises **arango.exceptions.IndexCreateError** – If create fails.

all(*skip: Optional[int] = None*, *limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return all documents in the collection.

Parameters

- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

checksum(*with_rev: bool = False*, *with_data: bool = False*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return collection checksum.

Parameters

- **with_rev** (*bool*) – Include document revisions in checksum calculation.
- **with_data** (*bool*) – Include document data in checksum calculation.

Returns Collection checksum.

Return type str

Raises **arango.exceptions.CollectionChecksumError** – If retrieval fails.

configure(*sync: Optional[bool] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Configure collection properties.

Parameters

- **sync** (*bool | None*) – Block until operations are synchronized to disk.
- **schema** (*dict*) – document schema for validation of objects.

Returns New collection properties.

Return type dict

Raises **arango.exceptions.CollectionConfigureError** – If operation fails.

conn
Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str

count() → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]
Return the total document count.

Returns Total document count.

Return type int

Raises ***arango.exceptions.DocumentCountError*** – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str

delete_index(*index_id: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete an index.

Parameters

- **index_id** (*str*) – Index ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing index.

Returns True if index was deleted successfully, False if index was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.IndexDeleteError*** – If delete fails.

delete_many(*documents: Sequence[Dict[str, Any]]*, *return_old: bool = False*, *check_rev: bool = True*, *sync: Optional[bool] = None*, *silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]
Delete multiple documents.

Note: If deleting a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were deleted successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single delete operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[str | dict]*) – Document IDs, keys or bodies. Document bodies must contain the "_id" or "_key" fields.
- **return_old** (*bool*) – Include bodies of the old documents in the result.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises ***arango.exceptions.DocumentDeleteError*** – If delete fails.

delete_match(*filters: Dict[str, Any], limit: Optional[int] = None, sync: Optional[bool] = None*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Delete matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single delete operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **limit** (*int | None*) – Max number of documents to delete. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Number of documents deleted.

Return type int

Raises ***arango.exceptions.DocumentDeleteError*** – If delete fails.

export(*limit: Optional[int] = None, count: bool = False, batch_size: Optional[int] = None, flush: bool = False, flush_wait: Optional[int] = None, ttl: Optional[numbers.Number] = None, filter_fields: Optional[Sequence[str]] = None, filter_type: str = 'include'*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Export all documents in the collection using a server cursor.

Parameters

- **flush** (*bool*) – If set to True, flush the write-ahead log prior to the export. If set to False, documents in the write-ahead log during the export are not included in the result.
- **flush_wait** (*int | None*) – Max wait time in seconds for write-ahead log flush.
- **count** (*bool*) – Include the document count in the server cursor.
- **batch_size** (*int | None*) – Max number of documents in the batch fetched by the cursor in one round trip.
- **limit** (*int | None*) – Max number of documents fetched by the cursor.
- **ttl** (*int | float | None*) – Time-to-live for the cursor on the server.
- **filter_fields** (*[str] | None*) – Document fields to filter with.
- **filter_type** (*str*) – Allowed values are "include" or "exclude".

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If export fails.

find(*filters: Dict[str, Any], skip: Optional[int] = None, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return all documents that match the given filters.

Parameters

- **filters** (*dict*) – Document filters.
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_by_text(*field: str, query: str, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents that match the given fulltext query.

Parameters

- **field** (*str*) – Document field with fulltext index.
- **query** (*str*) – Fulltext query.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_box(*latitude1: numbers.Number, longitude1: numbers.Number, latitude2: numbers.Number, longitude2: numbers.Number, skip: Optional[int] = None, limit: Optional[int] = None, index: Optional[str] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return all documents in an rectangular area.

Parameters

- **latitude1** (*int | float*) – First latitude.
- **longitude1** (*int | float*) – First longitude.
- **latitude2** (*int | float*) – Second latitude.
- **longitude2** (*int | float*) – Second longitude
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.
- **index** (*str | None*) – ID of the geo index to use (without the collection prefix). This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_radius(*latitude: numbers.Number*, *longitude: numbers.Number*, *radius: numbers.Number*, *distance_field: Optional[str] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents within a given radius around a coordinate.

A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.
- **radius** (*int | float*) – Max radius.
- **distance_field** (*str*) – Document field used to indicate the distance to the given coordinate. This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_range(*field: str*, *lower: int*, *upper: int*, *skip: Optional[int] = None*, *limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]

Return documents within a given range in a random order.

A skiplist index must be defined in the collection to use this method.

Parameters

- **field** (*str*) – Document field name.
- **lower** (*int*) – Lower bound (inclusive).
- **upper** (*int*) – Upper bound (exclusive).
- **skip** (*int | None*) – Number of documents to skip.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_near(*latitude: numbers.Number, longitude: numbers.Number, limit: Optional[int] = None*) → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return documents near a given coordinate.

Documents returned are sorted according to distance, with the nearest document being the first. If there are documents of equal distance, they are randomly chosen from the set until the limit is reached. A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.
- **limit** (*int | None*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

get_many(*documents: Sequence[Union[str, Dict[str, Any]]]*) → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return multiple documents ignoring any missing ones.

Parameters **documents** (*[str | dict]*) – List of document keys, IDs or bodies. Document bodies must contain the "_id" or "_key" fields.

Returns Documents. Missing ones are not included.

Return type [dict]

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

has(*document: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a document exists in the collection.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- **arango.exceptions.DocumentInError** – If check fails.
- **arango.exceptions.DocumentRevisionError** – If revisions mismatch.

ids() → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return the IDs of all documents in the collection.

Returns Document ID cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentIDsError*** – If retrieval fails.

import_bulk(*documents: Sequence[Dict[str, Any]], halt_on_error: bool = True, details: bool = True, from_prefix: Optional[str] = None, to_prefix: Optional[str] = None, overwrite: Optional[bool] = None, on_duplicate: Optional[str] = None, sync: Optional[bool] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Insert multiple documents into the collection.

Note: This method is faster than *arango.collection.Collection.insert_many()* but does not return as many details.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single insert operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.
- **halt_on_error** (*bool*) – Halt the entire import on an error.
- **details** (*bool*) – If set to True, the returned result will include an additional list of detailed error messages.
- **from_prefix** (*str*) – String prefix prepended to the value of "_from" field in each edge document inserted. For example, prefix "foo" prepended to "_from": "bar" will result in "_from": "foo/bar". Applies only to edge collections.
- **to_prefix** (*str*) – String prefix prepended to the value of "_to" field in edge document inserted. For example, prefix "foo" prepended to "_to": "bar" will result in "_to": "foo/bar". Applies only to edge collections.
- **overwrite** (*bool*) – If set to True, all existing documents are removed prior to the import. Indexes are still preserved.
- **on_duplicate** (*str*) – Action to take on unique key constraint violations (for documents with "_key" fields). Allowed values are "error" (do not import the new documents and count them as errors), "update" (update the existing documents while preserving any fields missing in the new ones), "replace" (replace the existing documents with new ones), and "ignore" (do not import the new documents and count them as ignored, as opposed to counting them as errors). Options "update" and "replace" may fail on secondary unique key constraint violations.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Result of the bulk import.

Return type dict

Raises *arango.exceptions.DocumentInsertError* – If import fails.

indexes() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the collection indexes.

Returns Collection indexes.

Return type [dict]

Raises *arango.exceptions.IndexListError* – If retrieval fails.

insert_many(*documents: Sequence[Dict[str, Any]], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]
Insert multiple documents.

Note: If inserting a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were inserted successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single insert operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include bodies of the new documents in the returned metadata. Ignored if parameter **silent** is set to True
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate keys and the existing documents are replaced.
- **return_old** (*bool*) – Include body of the old documents if replaced. Applies only when value of **overwrite** is set to True.

Returns List of document metadata (e.g. document keys, revisions) and any exception, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

keys() → Union[arango.cursor.Cursor, arango.job.AsyncJob[arango.cursor.Cursor][arango.cursor.Cursor], arango.job.BatchJob[arango.cursor.Cursor][arango.cursor.Cursor], None]
Return the keys of all documents in the collection.

Returns Document key cursor.

Return type *arango.cursor.Cursor*

Raises *arango.exceptions.DocumentKeysError* – If retrieval fails.

load() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Load the collection into memory.

Returns True if collection was loaded successfully.

Return type bool

Raises *arango.exceptions.CollectionLoadError* – If operation fails.

load_indexes() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Cache all indexes in the collection into memory.

Returns True if index was loaded successfully.

Return type bool

Raises *arango.exceptions.IndexLoadError* – If operation fails.

name
Return collection name.

Returns Collection name.

Return type str

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return collection properties.

Returns Collection properties.

Return type dict

Raises *arango.exceptions.CollectionPropertiesError* – If retrieval fails.

random() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return a random document from the collection.

Returns A random document.

Return type dict

Raises *arango.exceptions.DocumentGetError* – If retrieval fails.

recalculate_count() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Recalculate the document count.

Returns True if recalculation was successful.

Return type bool

Raises *arango.exceptions.CollectionRecalculateCountError* – If operation fails.

rename(*new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Rename the collection.

Renames may not be reflected immediately in async execution, batch execution or transactions. It is recommended to initialize new API wrappers after a rename.

Parameters **new_name** (*str*) – New collection name.

Returns True if collection was renamed successfully.

Return type bool

Raises *arango.exceptions.CollectionRenameError* – If rename fails.

replace_many(*documents: Sequence[Dict[str, Any]], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]
Replace multiple documents.

Note: If replacing a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were replaced successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single replace operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – New documents to replace the old ones with. They must contain the "_id" or "_key" fields. Edge documents must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoServerError] | bool

Raises *arango.exceptions.DocumentReplaceError* – If replace fails.

replace_match(*filters: Dict[str, Any], body: Dict[str, Any], limit: Optional[int] = None, sync: Optional[bool] = None*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]
Replace matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single replace operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – New document body.
- **limit** (*int | None*) – Max number of documents to replace. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.

Returns Number of documents replaced.

Return type int

Raises ***arango.exceptions.DocumentReplaceError*** – If replace fails.

responsible_shard(*document: Dict[str, Any]*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return the ID of the shard responsible for given **document**.

If the document does not exist, return the shard that would be responsible.

Returns Shard ID

Return type str

revision() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return collection revision.

Returns Collection revision.

Return type str

Raises ***arango.exceptions.CollectionRevisionError*** – If retrieval fails.

statistics() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return collection statistics.

Returns Collection statistics.

Return type dict

Raises ***arango.exceptions.CollectionStatisticsError*** – If retrieval fails.

truncate() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete all documents in the collection.

Returns True if collection was truncated successfully.

Return type bool

Raises ***arango.exceptions.CollectionTruncateError*** – If operation fails.

unload() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Unload the collection from memory.

Returns True if collection was unloaded successfully.

Return type bool

Raises ***arango.exceptions.CollectionUnloadError*** – If operation fails.

update_many(*documents: Sequence[Dict[str, Any]], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]], arango.job.AsyncJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], arango.job.BatchJob[typing.Union[bool, typing.List[typing.Union[typing.Dict[str, typing.Any], arango.exceptions.ArangoServerError]]]][Union[bool, List[Union[Dict[str, Any], arango.exceptions.ArangoServerError]]]], None]
Update multiple documents.

Note: If updating a document fails, the exception is not raised but returned as an object in the result list. It is up to you to inspect the list to determine which documents were updated successfully (returns document metadata) and which were not (returns exception object).

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single update operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **documents** (*[dict]*) – Partial or full documents with the updated values. They must contain the "_id" or "_key" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises *arango.exceptions.DocumentUpdateError* – If update fails.

update_match(*filters: Dict[str, Any], body: Dict[str, Any], limit: Optional[int] = None, keep_none: bool = True, sync: Optional[bool] = None, merge: bool = True*) → Union[int, arango.job.AsyncJob[int][int], arango.job.BatchJob[int][int], None]

Update matching documents.

Note: In edge/vertex collections, this method does NOT provide the transactional guarantees and validations that single update operation does for graphs. If these properties are required, see *arango.database.StandardDatabase.begin_batch_execution()* for an alternative approach.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – Full or partial document body with the updates.
- **limit** (*int | None*) – Max number of documents to update. If the limit is lower than the number of matched documents, random documents are chosen. This parameter is not supported on sharded collections.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.

Returns Number of documents updated.

Return type int

Raises *arango.exceptions.DocumentUpdateError* – If update fails.

username

Return the username.

Returns Username.

Return type str

3.33.22 StandardDatabase

class arango.database.**StandardDatabase**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection]*)

Standard database API wrapper.

begin_async_execution(*return_result: bool = True*) → arango.database.AsyncDatabase

Begin async execution.

Parameters **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.AsyncJob*, which you can use to retrieve results from server once available. If set to False, API executions return None and no results are stored on server.

Returns Database API wrapper object specifically for async execution.

Return type *arango.database.AsyncDatabase*

begin_batch_execution(*return_result: bool = True*) → arango.database.BatchDatabase
Begin batch execution.

Parameters **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.BatchJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.

Returns Database API wrapper object specifically for batch execution.

Return type *arango.database.BatchDatabase*

begin_transaction(*read: Union[str, Sequence[str], None] = None, write: Union[str, Sequence[str], None] = None, exclusive: Union[str, Sequence[str], None] = None, sync: Optional[bool] = None, allow_implicit: Optional[bool] = None, lock_timeout: Optional[int] = None, max_size: Optional[int] = None*) → arango.database.TransactionDatabase
Begin a transaction.

Parameters

- **read** (*str | [str] | None*) – Name(s) of collections read during transaction. Read-only collections are added lazily but should be declared if possible to avoid deadlocks.
- **write** (*str | [str] | None*) – Name(s) of collections written to during transaction with shared access.
- **exclusive** (*str | [str] | None*) – Name(s) of collections written to during transaction with exclusive access.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **allow_implicit** (*bool | None*) – Allow reading from undeclared collections.
- **lock_timeout** (*int | None*) – Timeout for waiting on collection locks. If not given, a default value is used. Setting it to 0 disables the timeout.
- **max_size** (*int | None*) – Max transaction size in bytes.

Returns Database API wrapper object specifically for transactions.

Return type *arango.database.TransactionDatabase*

analyzer(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return analyzer details.

Parameters **name** (*str*) – Analyzer name.

Returns Analyzer details.

Return type dict

Raises ***arango.exceptions.AnalyzerGetError*** – If retrieval fails.

analyzers() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of analyzers.

Returns List of analyzers.

Return type [dict]

Raises ***arango.exceptions.AnalyzerListError*** – If retrieval fails.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status: str*, *count: Optional[int] = None*) → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]
Return IDs of async jobs with given status.

Parameters

- **status** (*str*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str]

Raises ***arango.exceptions.AsyncJobListError*** – If retrieval fails.

backup
Return Backup API wrapper.

Returns Backup API wrapper.

Return type *arango.backup.Backup*

clear_async_jobs(*threshold: Optional[int] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int | None*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

cluster
Return Cluster API wrapper.

Returns Cluster API wrapper.

Return type *arango.cluster.Cluster*

collection(*name: str*) → arango.collection.StandardCollection
Return the standard collection API wrapper.

Parameters **name** (*str*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises *arango.exceptions.CollectionListError* – If retrieval fails.

conn
Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str

create_analyzer(*name: str*, *analyzer_type: str*, *properties: Optional[Dict[str, Any]] = None*, *features: Optional[Sequence[str]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **analyzer_type** (*str*) – Analyzer type.
- **properties** (*dict | None*) – Analyzer properties.
- **features** (*list | None*) – Analyzer features.

Returns Analyzer details.

Return type dict

Raises *arango.exceptions.AnalyzerCreateError* – If create fails.

create_arangosearch_view(*name: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Create an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict | None*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

create_collection(*name: str*, *sync: bool = False*, *system: bool = False*, *edge: bool = False*, *user_keys: bool = True*, *key_increment: Optional[int] = None*, *key_offset: Optional[int] = None*, *key_generator: str = 'traditional'*, *shard_fields: Optional[Sequence[str]] = None*, *shard_count: Optional[int] = None*, *replication_factor: Optional[int] = None*, *shard_like: Optional[str] = None*, *sync_replication: Optional[bool] = None*, *enforce_replication_factor: Optional[bool] = None*, *sharding_strategy: Optional[str] = None*, *smart_join_attribute: Optional[str] = None*, *write_concern: Optional[int] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[arango.collection.StandardCollection, arango.job.AsyncJob[arango.collection.StandardCollection][arango.collection.StandardCollection], arango.job.BatchJob[arango.collection.StandardCollection][arango.collection.StandardCollection], None]

Create a new collection.

Parameters

- **name** (*str*) – Collection name.
- **sync** (*bool | None*) – If set to True, document operations via the collection will block until synchronized to disk by default.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **key_generator** (*str*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.
- **sharding_strategy** (*str*) – Sharding strategy. Available for ArangoDB version and up only. Possible values are "community-compat", "enterprise-compat", "enterprise-

smart-edge-compat", "hash" and "enterprise-hash-smart-edge". Refer to ArangoDB documentation for more details on each value.

- **smart_join_attribute** (*str*) – Attribute of the collection which must contain the shard key value of the smart join collection. The shard key for the documents must contain the value of this attribute, followed by a colon ":" and the primary key of the document. Requires parameter **shard_like** to be set to the name of another collection, and parameter **shard_fields** to be set to a single shard key attribute, with another colon ":" at the end. Available only for enterprise version of ArangoDB.
- **write_concern** (*int*) – Write concern for the collection. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time. The value of this parameter cannot be larger than that of **replication_factor**. Default value is 1. Used for clusters only.
- **schema** (*dict*) – Optional dict specifying the collection level schema for documents. See ArangoDB documentation for more information on document schema validation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name: str*, *users: Optional[Sequence[Dict[str, Any]]] = None*, *replication_factor: Union[int, str, None] = None*, *write_concern: Optional[int] = None*, *sharding: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Create a new database.

Parameters

- **name** (*str*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.
- **replication_factor** (*int | str*) – Default replication factor for collections created in this database. Special values include "satellite" which replicates the collection to every DBServer, and 1 which disables replication. Used for clusters only.
- **write_concern** (*int*) – Default write concern for collections created in this database. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time, however. Value of this parameter can not be larger than the value of **replication_factor**. Used for clusters only.
- **sharding** (*str*) – Sharding method used for new collections in this database. Allowed values are: "", "flexible" and "single". The first two are equivalent. Used for clusters only.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name: str*, *edge_definitions: Optional[Sequence[Dict[str, Any]]] = None*, *orphan_collections: Optional[Sequence[str]] = None*, *smart: Optional[bool] = None*, *smart_field: Optional[str] = None*, *shard_count: Optional[int] = None*) → Union[arango.graph.Graph, arango.job.AsyncJob[arango.graph.Graph][arango.graph.Graph], arango.job.BatchJob[arango.graph.Graph][arango.graph.Graph], None]

Create a new graph.

Parameters

- **name** (*str*) – Graph name.
- **edge_definitions** (*[dict] | None*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str] | None*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool | None*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | None*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int | None*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name: str*, *command: str*, *params: Optional[Dict[str, Any]] = None*, *period: Optional[int] = None*, *offset: Optional[int] = None*, *task_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new server task.

Parameters

- **name** (*str*) – Name of the server task.

- **command** (*str*) – Javascript command to execute.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **period** (*int | None*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int | None*) – Initial delay before execution in seconds.
- **task_id** (*str | None*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises *arango.exceptions.TaskCreateError* – If create fails.

create_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – Password.
- **active** (*bool | None*) – True if user is active, False otherwise.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name: str*, *view_type: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a view.

Parameters

- **name** (*str*) – View name.
- **view_type** (*str*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return the names all databases.

Returns Database names.

Return type [str]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str

delete_analyzer(*name: str*, *force: bool = False*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **force** (*bool*) – Remove the analyzer configuration even if in use.
- **ignore_missing** (*bool*) – Do not raise an exception on missing analyzer.

Returns True if analyzer was deleted successfully, False if analyzer was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.AnalyzerDeleteError* – If delete fails.

delete_collection(*name: str*, *ignore_missing: bool = False*, *system: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete the collection.

Parameters

- **name** (*str*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete the database.

Parameters

- **name** (*str*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name: str*, *ignore_missing: bool = False*, *drop_collections: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Drop the graph of the given name from the database.

Parameters

- **name** (*str*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool | None*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a server task.

Parameters

- **task_id** (*str*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a user.

Parameters

- **username** (*str*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.UserDeleteError*** – If delete fails.

delete_view(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a view.

Parameters

- **name** (*str*) – View name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing view.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.ViewDeleteError*** – If delete fails.

details() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return ArangoDB server details.

Returns Server details.

Return type dict

Raises ***arango.exceptions.ServerDetailsError*** – If retrieval fails.

document(*document: Dict[str, Any]*, *rev: Optional[str] = None*, *check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return a document.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.

- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

echo() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises ***arango.exceptions.ServerEchoError*** – If retrieval fails.

encryption() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Rotate the user-supplied keys for encryption.

This method is available only for enterprise edition of ArangoDB.

Returns New TLS data.

Return type dict

Raises ***arango.exceptions.ServerEncryptionError*** – If retrieval fails.

engine() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the database engine details.

Returns Database engine details.

Return type dict

Raises ***arango.exceptions.ServerEngineError*** – If retrieval fails.

execute_transaction(*command: str*, *params: Optional[Dict[str, Any]] = None*, *read: Optional[Sequence[str]] = None*, *write: Optional[Sequence[str]] = None*, *sync: Optional[bool] = None*, *timeout: Optional[numbers.Number] = None*, *max_size: Optional[int] = None*, *allow_implicit: Optional[bool] = None*, *intermediate_commit_count: Optional[int] = None*, *intermediate_commit_size: Optional[int] = None*) → Union[Any, arango.job.AsyncJob[typing.Any][Any], arango.job.BatchJob[typing.Any][Any], None]
Execute raw Javascript command in transaction.

Parameters

- **command** (*str*) – Javascript command to execute.
- **read** (*[str] | None*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str] | None*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.

- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **timeout** (*int | None*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int | None*) – Max transaction size limit in bytes.
- **allow_implicit** (*bool | None*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int | None*) – Max number of operations after which an intermediate commit is performed automatically.
- **intermediate_commit_size** (*int | None*) – Max size of operations in bytes after which an intermediate commit is performed automatically.

Returns Return value of **command**.

Return type Any

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name: str*) → arango.graph.Graph
Return the graph API wrapper.

Parameters **name** (*str*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if collection exists in the database.

Parameters **name** (*str*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a database exists.

Parameters **name** (*str*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document: Dict[str, Any], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if a document exists.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if a graph exists in the database.

Parameters **name** (*str*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Check if user exists.

Parameters **username** (*str*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection: str, document: Dict[str, Any], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False, overwrite_mode: Optional[str] = None, keep_none: Optional[bool] = None, merge: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new document.

Parameters

- **collection** (*str*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.
- **overwrite_mode** (*str | None*) – Overwrite behavior used when the document key exists already. Allowed values are "replace" (replace-insert) or "update" (update-insert). Implicitly sets the value of parameter **overwrite**.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely. Applies only when **overwrite_mode** is set to "update" (update-insert).
- **merge** (*bool | None*) – If set to True (default), sub-dictionaries are merged instead of the new one overwriting the old one. Applies only when **overwrite_mode** is set to "update" (update-insert).

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return information on currently loaded JWT secrets.

Returns Information on currently loaded JWT secrets.

Return type dict

log_levels() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return current logging levels.

Returns Current logging levels.

Return type dict

metrics() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server metrics in Prometheus format.

Returns Server metrics in Prometheus format.

Return type str

name
Return database name.

Returns Database name.

Return type str

permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns Permission for given database or collection.

Return type str

Raises *arango.exceptions.PermissionGetError* – If retrieval fails.

permissions(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return user permissions for all databases and collections.

Parameters **username** (*str*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raises *arango.exceptions.PermissionListError* – If retrieval fails.

pregel

Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return database properties.

Returns Database properties.

Return type dict

Raises *arango.exceptions.DatabasePropertiesError* – If retrieval fails.

read_log(*upto: Union[str, int, None] = None*, *level: Union[str, int, None] = None*, *start: Optional[int] = None*, *size: Optional[int] = None*, *offset: Optional[int] = None*, *search: Optional[str] = None*, *sort: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Read the global log from server.

Parameters

- **upto** (*int | str*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*int | str*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str*) – Return only the log entries containing the given text.

- **sort** (*str*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises *arango.exceptions.ServerReadLogError* – If read fails.

reload_jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Hot-reload JWT secrets.

Calling this without payload reloads JWT secrets from disk. Only files specified via arangod startup option `--server.jwt-secret-keyfile` or `--server.jwt-secret-folder` are used. It is not possible to change the location where files are loaded from without restarting the server.

Returns Information on reloaded JWT secrets.

Return type dict

reload_routing() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises *arango.exceptions.ServerReloadRoutingError* – If reload fails.

reload_tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Reload TLS data (server key, client-auth CA).

Returns New TLS data.

Return type dict

rename_view(*name: str*, *new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Rename a view.

Parameters

- **name** (*str*) – View name.
- **new_name** (*str*) – New view name.

Returns True if view was renamed successfully.

Return type bool

Raises *arango.exceptions.ViewRenameError* – If delete fails.

replace_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace an ArangoSearch view.

Parameters

- **name** (*str*) – View name.

- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replace_document(*document: Dict[str, Any], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username: str, password: str, active: Optional[bool] = None, extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Replace a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replication
Return Replication API wrapper.

Returns Replication API wrapper.

Return type *arango.replication.Replication*

required_db_version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return required version of target database.

Returns Required version of target database.

Return type str

Raises ***arango.exceptions.ServerRequiredDBVersionError*** – If retrieval fails.

reset_permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reset user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server role.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY", "AGENT" (Agency node in a cluster) or "UNDEFINED".

Return type str

Raises ***arango.exceptions.ServerRoleError*** – If retrieval fails.

run_tests(*tests: Sequence[str]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Run available unittests on the server.

Parameters **tests** (*[str]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises ***arango.exceptions.ServerRunTestsError*** – If execution fails.

set_log_levels(**kwargs) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises ***arango.exceptions.ServerShutdownError*** – If shutdown fails.

statistics(*description: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return server statistics.

Returns Server statistics.

Return type dict

Raises ***arango.exceptions.ServerStatisticsError*** – If retrieval fails.

status() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises ***arango.exceptions.ServerStatusError*** – If retrieval fails.

task(*task_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the details of an active server task.

Parameters **task_id** (*str*) – Server task ID.

Returns Server task details.

Return type dict

Raises *arango.exceptions.TaskGetError* – If retrieval fails.

tasks() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises *arango.exceptions.TaskListError* – If retrieval fails.

time() → Union[datetime.datetime, arango.job.AsyncJob[datetime.datetime][datetime.datetime], arango.job.BatchJob[datetime.datetime][datetime.datetime], None]
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return TLS data (server key, client-auth CA).

Returns TLS data.

Return type dict

update_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

update_document(*document: Dict[str, Any], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username: str*, *permission: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Update user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **permission** (*str*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserUpdateError* – If update fails.

update_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user details.

Parameters **username** (*str*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str

users() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return ArangoDB server version.

Returns Server version.

Return type str

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of views and their summaries.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.33.23 TransactionDatabase

class arango.database.**TransactionDatabase**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], read: Union[str, Sequence[str], None] = None, write: Union[str, Sequence[str], None] = None, exclusive: Union[str, Sequence[str], None] = None, sync: Optional[bool] = None, allow_implicit: Optional[bool] = None, lock_timeout: Optional[int] = None, max_size: Optional[int] = None*)
Database API wrapper tailored specifically for transactions.

See *arango.database.StandardDatabase.begin_transaction()*.

Parameters

- **connection** – HTTP connection.
- **read** (*str | [str] | None*) – Name(s) of collections read during transaction. Read-only collections are added lazily but should be declared if possible to avoid deadlocks.
- **write** (*str | [str] | None*) – Name(s) of collections written to during transaction with shared access.
- **exclusive** (*str | [str] | None*) – Name(s) of collections written to during transaction with exclusive access.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **allow_implicit** (*bool | None*) – Allow reading from undeclared collections.
- **lock_timeout** (*int | None*) – Timeout for waiting on collection locks. If not given, a default value is used. Setting it to 0 disables the timeout.
- **max_size** (*int | None*) – Max transaction size in bytes.

transaction_id
Return the transaction ID.

Returns Transaction ID.

Return type str

transaction_status() → str
Return the transaction status.

Returns Transaction status.

Return type str

Raises *arango.exceptions.TransactionStatusError* – If retrieval fails.

commit_transaction() → bool
Commit the transaction.

Returns True if commit was successful.

Return type bool

Raises *arango.exceptions.TransactionCommitError* – If commit fails.

abort_transaction() → bool
Abort the transaction.

Returns True if the abort operation was successful.

Return type bool

Raises *arango.exceptions.TransactionAbortError* – If abort fails.

analyzer(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return analyzer details.

Parameters **name** (*str*) – Analyzer name.

Returns Analyzer details.

Return type dict

Raises *arango.exceptions.AnalyzerGetError* – If retrieval fails.

analyzers() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of analyzers.

Returns List of analyzers.

Return type [dict]

Raises *arango.exceptions.AnalyzerListError* – If retrieval fails.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status: str*, *count: Optional[int] = None*) → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]
Return IDs of async jobs with given status.

Parameters

- **status** (*str*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str]

Raises *arango.exceptions.AsyncJobListError* – If retrieval fails.

backup
Return Backup API wrapper.

Returns Backup API wrapper.

Return type *arango.backup.Backup*

clear_async_jobs(*threshold: Optional[int] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int | None*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises *arango.exceptions.AsyncJobClearError* – If operation fails.

cluster
Return Cluster API wrapper.

Returns Cluster API wrapper.

Return type *arango.cluster.Cluster*

collection(*name: str*) → arango.collection.StandardCollection
Return the standard collection API wrapper.

Parameters **name** (*str*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises *arango.exceptions.CollectionListError* – If retrieval fails.

conn
Return the HTTP connection.

Returns HTTP connection.

Return type arango.connection.BasicConnection | arango.connection.JwtConnection | arango.connection.JwtSuperuserConnection

context
Return the API execution context.

Returns API execution context. Possible values are “default”, “async”, “batch” and “transaction”.

Return type str

create_analyzer(*name: str*, *analyzer_type: str*, *properties: Optional[Dict[str, Any]] = None*, *features: Optional[Sequence[str]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **analyzer_type** (*str*) – Analyzer type.
- **properties** (*dict | None*) – Analyzer properties.
- **features** (*list | None*) – Analyzer features.

Returns Analyzer details.

Return type dict

Raises ***arango.exceptions.AnalyzerCreateError*** – If create fails.

create_arangosearch_view(*name: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict | None*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewCreateError*** – If create fails.

create_collection(*name: str*, *sync: bool = False*, *system: bool = False*, *edge: bool = False*, *user_keys: bool = True*, *key_increment: Optional[int] = None*, *key_offset: Optional[int] = None*, *key_generator: str = 'traditional'*, *shard_fields: Optional[Sequence[str]] = None*, *shard_count: Optional[int] = None*, *replication_factor: Optional[int] = None*, *shard_like: Optional[str] = None*, *sync_replication: Optional[bool] = None*, *enforce_replication_factor: Optional[bool] = None*, *sharding_strategy: Optional[str] = None*, *smart_join_attribute: Optional[str] = None*, *write_concern: Optional[int] = None*, *schema: Optional[Dict[str, Any]] = None*) → Union[arango.collection.StandardCollection, arango.job.AsyncJob[arango.collection.StandardCollection][arango.collection.StandardCollection], arango.job.BatchJob[arango.collection.StandardCollection][arango.collection.StandardCollection], None]

Create a new collection.

Parameters

- **name** (*str*) – Collection name.
- **sync** (*bool | None*) – If set to True, document operations via the collection will block until synchronized to disk by default.

- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **key_generator** (*str*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.
- **sharding_strategy** (*str*) – Sharding strategy. Available for ArangoDB version and up only. Possible values are "community-compat", "enterprise-compat", "enterprise-smart-edge-compat", "hash" and "enterprise-hash-smart-edge". Refer to ArangoDB documentation for more details on each value.
- **smart_join_attribute** (*str*) – Attribute of the collection which must contain the shard key value of the smart join collection. The shard key for the documents must contain the value of this attribute, followed by a colon ":" and the primary key of the document. Requires parameter **shard_like** to be set to the name of another collection, and parameter **shard_fields** to be set to a single shard key attribute, with another colon ":" at the end. Available only for enterprise version of ArangoDB.
- **write_concern** (*int*) – Write concern for the collection. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time. The value of this parameter cannot be larger than that of **replication_factor**. Default value is 1. Used for clusters only.
- **schema** (*dict*) – Optional dict specifying the collection level schema for documents. See ArangoDB documentation for more information on document schema validation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises *arango.exceptions.CollectionCreateError* – If create fails.

create_database(*name: str*, *users: Optional[Sequence[Dict[str, Any]]] = None*, *replication_factor: Union[int, str, None] = None*, *write_concern: Optional[int] = None*, *sharding: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Create a new database.

Parameters

- **name** (*str*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.
- **replication_factor** (*int | str*) – Default replication factor for collections created in this database. Special values include "satellite" which replicates the collection to every DBServer, and 1 which disables replication. Used for clusters only.
- **write_concern** (*int*) – Default write concern for collections created in this database. Determines how many copies of each shard are required to be in sync on different DBServers. If there are less than these many copies in the cluster a shard will refuse to write. Writes to shards with enough up-to-date copies will succeed at the same time, however. Value of this parameter can not be larger than the value of **replication_factor**. Used for clusters only.
- **sharding** (*str*) – Sharding method used for new collections in this database. Allowed values are: "", "flexible" and "single". The first two are equivalent. Used for clusters only.

Returns True if database was created successfully.

Return type bool

Raises *arango.exceptions.DatabaseCreateError* – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name: str*, *edge_definitions: Optional[Sequence[Dict[str, Any]]] = None*, *orphan_collections: Optional[Sequence[str]] = None*, *smart: Optional[bool] = None*, *smart_field: Optional[str] = None*, *shard_count: Optional[int] = None*) → Union[arango.graph.Graph, arango.job.AsyncJob[arango.graph.Graph][arango.graph.Graph], arango.job.BatchJob[arango.graph.Graph][arango.graph.Graph], None]

Create a new graph.

Parameters

- **name** (*str*) – Graph name.
- **edge_definitions** (*[dict] | None*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).

- **orphan_collections** (*[str] | None*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool | None*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | None*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int | None*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name: str*, *command: str*, *params: Optional[Dict[str, Any]] = None*, *period: Optional[int] = None*, *offset: Optional[int] = None*, *task_id: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new server task.

Parameters

- **name** (*str*) – Name of the server task.
- **command** (*str*) – Javascript command to execute.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **period** (*int | None*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int | None*) – Initial delay before execution in seconds.
- **task_id** (*str | None*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises ***arango.exceptions.TaskCreateError*** – If create fails.

create_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a new user.

Parameters

- **username** (*str*) – Username.

- **password** (*str | None*) – Password.
- **active** (*bool | None*) – True if user is active, False otherwise.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name: str*, *view_type: str*, *properties: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Create a view.

Parameters

- **name** (*str*) – View name.
- **view_type** (*str*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases() → Union[List[str], arango.job.AsyncJob[typing.List[str]][List[str]], arango.job.BatchJob[typing.List[str]][List[str]], None]

Return the names all databases.

Returns Database names.

Return type [str]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name

Return the name of the current database.

Returns Database name.

Return type str

delete_analyzer(*name: str*, *force: bool = False*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete an analyzer.

Parameters

- **name** (*str*) – Analyzer name.
- **force** (*bool*) – Remove the analyzer configuration even if in use.
- **ignore_missing** (*bool*) – Do not raise an exception on missing analyzer.

Returns True if analyzer was deleted successfully, False if analyzer was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.AnalyzerDeleteError* – If delete fails.

delete_collection(*name: str*, *ignore_missing: bool = False*, *system: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete the collection.

Parameters

- **name** (*str*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.CollectionDeleteError*** – If delete fails.

delete_database(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete the database.

Parameters

- **name** (*str*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.DatabaseDeleteError*** – If delete fails.

delete_document(*document: Union[str, Dict[str, Any]]*, *rev: Optional[str] = None*, *check_rev: bool = True*, *ignore_missing: bool = False*, *return_old: bool = False*, *sync: Optional[bool] = None*, *silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a document.

Parameters

- **document** (*str | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name: str*, *ignore_missing: bool = False*, *drop_collections: Optional[bool] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Drop the graph of the given name from the database.

Parameters

- **name** (*str*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool | None*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a server task.

Parameters

- **task_id** (*str*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Delete a user.

Parameters

- **username** (*str*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.UserDeleteError*** – If delete fails.

delete_view(*name: str*, *ignore_missing: bool = False*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Delete a view.

Parameters

- **name** (*str*) – View name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing view.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.ViewDeleteError*** – If delete fails.

details() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises ***arango.exceptions.ServerDetailsError*** – If retrieval fails.

document(*document: Dict[str, Any]*, *rev: Optional[str] = None*, *check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]
Return a document.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

echo() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises ***arango.exceptions.ServerEchoError*** – If retrieval fails.

encryption() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Rotate the user-supplied keys for encryption.

This method is available only for enterprise edition of ArangoDB.

Returns New TLS data.

Return type dict

Raises *arango.exceptions.ServerEncryptionError* – If retrieval fails.

engine() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the database engine details.

Returns Database engine details.

Return type dict

Raises *arango.exceptions.ServerEngineError* – If retrieval fails.

execute_transaction(*command: str*, *params: Optional[Dict[str, Any]] = None*, *read: Optional[Sequence[str]] = None*, *write: Optional[Sequence[str]] = None*, *sync: Optional[bool] = None*, *timeout: Optional[numbers.Number] = None*, *max_size: Optional[int] = None*, *allow_implicit: Optional[bool] = None*, *intermediate_commit_count: Optional[int] = None*, *intermediate_commit_size: Optional[int] = None*) → Union[Any, arango.job.AsyncJob[typing.Any][Any], arango.job.BatchJob[typing.Any][Any], None]
Execute raw Javascript command in transaction.

Parameters

- **command** (*str*) – Javascript command to execute.
- **read** (*[str] | None*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str] | None*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict | None*) – Optional parameters passed into the Javascript command.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **timeout** (*int | None*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int | None*) – Max transaction size limit in bytes.
- **allow_implicit** (*bool | None*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int | None*) – Max number of operations after which an intermediate commit is performed automatically.
- **intermediate_commit_size** (*int | None*) – Max size of operations in bytes after which an intermediate commit is performed automatically.

Returns Return value of **command**.

Return type Any

Raises *arango.exceptions.TransactionExecuteError* – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph (*name: str*) → arango.graph.Graph
Return the graph API wrapper.

Parameters **name** (*str*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs () → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection (*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if collection exists in the database.

Parameters **name** (*str*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database (*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a database exists.

Parameters **name** (*str*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document (*document: Dict[str, Any], rev: Optional[str] = None, check_rev: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a document exists.

Parameters

- **document** (*str | dict*) – Document ID or body with "_id" field.
- **rev** (*str | None*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if a graph exists in the database.

Parameters **name** (*str*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Check if user exists.

Parameters **username** (*str*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection: str, document: Dict[str, Any], return_new: bool = False, sync: Optional[bool] = None, silent: bool = False, overwrite: bool = False, return_old: bool = False, overwrite_mode: Optional[str] = None, keep_none: Optional[bool] = None, merge: Optional[bool] = None*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Insert a new document.

Parameters

- **collection** (*str*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.
- **overwrite_mode** (*str | None*) – Overwrite behavior used when the document key exists already. Allowed values are "replace" (replace-insert) or "update" (update-insert). Implicitly sets the value of parameter **overwrite**.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely. Applies only when **overwrite_mode** is set to "update" (update-insert).
- **merge** (*bool | None*) – If set to True (default), sub-dictionaries are merged instead of the new one overwriting the old one. Applies only when **overwrite_mode** is set to "update" (update-insert).

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return information on currently loaded JWT secrets.

Returns Information on currently loaded JWT secrets.

Return type dict

log_levels() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return current logging levels.

Returns Current logging levels.

Return type dict

metrics() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server metrics in Prometheus format.

Returns Server metrics in Prometheus format.

Return type str

name
Return database name.

Returns Database name.

Return type str

permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns Permission for given database or collection.

Return type str

Raises *arango.exceptions.PermissionGetError* – If retrieval fails.

permissions(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user permissions for all databases and collections.

Parameters **username** (*str*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raises *arango.exceptions.PermissionListError* – If retrieval fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return database properties.

Returns Database properties.

Return type dict

Raises ***arango.exceptions.DatabasePropertiesError*** – If retrieval fails.

read_log(*upto: Union[str, int, None] = None*, *level: Union[str, int, None] = None*, *start: Optional[int] = None*, *size: Optional[int] = None*, *offset: Optional[int] = None*, *search: Optional[str] = None*, *sort: Optional[str] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Read the global log from server.

Parameters

- **upto** (*int | str*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*int | str*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str*) – Return only the log entries containing the given text.
- **sort** (*str*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises ***arango.exceptions.ServerReadLogError*** – If read fails.

reload_jwt_secrets() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Hot-reload JWT secrets.

Calling this without payload reloads JWT secrets from disk. Only files specified via arangod startup option `--server.jwt-secret-keyfile` or `--server.jwt-secret-folder` are used. It is not possible to change the location where files are loaded from without restarting the server.

Returns Information on reloaded JWT secrets.

Return type dict

reload_routing() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises *arango.exceptions.ServerReloadRoutingError* – If reload fails.

reload_tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Reload TLS data (server key, client-auth CA).

Returns New TLS data.

Return type dict

rename_view(*name: str*, *new_name: str*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Rename a view.

Parameters

- **name** (*str*) – View name.
- **new_name** (*str*) – New view name.

Returns True if view was renamed successfully.

Return type bool

Raises *arango.exceptions.ViewRenameError* – If delete fails.

replace_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Replace an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewReplaceError* – If replace fails.

replace_document(*document: Dict[str, Any], check_rev: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.

- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username: str*, *password: str*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Replace a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

replication

Return Replication API wrapper.

Returns Replication API wrapper.

Return type *arango.replication.Replication*

required_db_version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return required version of target database.

Returns Required version of target database.

Return type str

Raises ***arango.exceptions.ServerRequiredDBVersionError*** – If retrieval fails.

reset_permission(*username: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Reset user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **database** (*str*) – Database name.
- **collection** (*str*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return server role.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY", "AGENT" (Agency node in a cluster) or "UNDEFINED".

Return type str

Raises ***arango.exceptions.ServerRoleError*** – If retrieval fails.

run_tests(*tests: Sequence[str]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Run available unittests on the server.

Parameters **tests** (*[str]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises ***arango.exceptions.ServerRunTestsError*** – If execution fails.

set_log_levels(***kwargs*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
```

(continues on next page)

(continued from previous page)

```
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown() → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises ***arango.exceptions.ServerShutdownError*** – If shutdown fails.

statistics(*description: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return server statistics.

Returns Server statistics.

Return type dict

Raises ***arango.exceptions.ServerStatisticsError*** – If retrieval fails.

status() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises ***arango.exceptions.ServerStatusError*** – If retrieval fails.

task(*task_id: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the details of an active server task.

Parameters **task_id** (*str*) – Server task ID.

Returns Server task details.

Return type dict

Raises ***arango.exceptions.TaskGetError*** – If retrieval fails.

tasks() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises ***arango.exceptions.TaskListError*** – If retrieval fails.

time() → Union[datetime.datetime, arango.job.AsyncJob[datetime.datetime][datetime.datetime], arango.job.BatchJob[datetime.datetime][datetime.datetime], None]
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

tls() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return TLS data (server key, client-auth CA).

Returns TLS data.

Return type dict

update_arangosearch_view(*name: str, properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Update an ArangoSearch view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

update_document(*document: Dict[str, Any], check_rev: bool = True, merge: bool = True, keep_none: bool = True, return_new: bool = False, return_old: bool = False, sync: Optional[bool] = None, silent: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool | None*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username: str*, *permission: str*, *database: str*, *collection: Optional[str] = None*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]

Update user permission for a specific database or collection.

Parameters

- **username** (*str*) – Username.
- **permission** (*str*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).
- **database** (*str*) – Database name.
- **collection** (*str | None*) – Collection name.

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username: str*, *password: Optional[str] = None*, *active: Optional[bool] = None*, *extra: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a user.

Parameters

- **username** (*str*) – Username.
- **password** (*str | None*) – New password.
- **active** (*bool | None*) – Whether the user is active.
- **extra** (*dict | None*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name: str*, *properties: Dict[str, Any]*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Update a view.

Parameters

- **name** (*str*) – View name.
- **properties** (*dict*) – View properties. For more information see https://www.arangodb.com/docs/stable/http/views-arangosearch.html

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return user details.

Parameters **username** (*str*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str

users() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version() → Union[str, arango.job.AsyncJob[str][str], arango.job.BatchJob[str][str], None]
Return ArangoDB server version.

Returns Server version.

Return type str

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name: str*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views() → Union[List[Dict[str, Any]], arango.job.AsyncJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], arango.job.BatchJob[typing.List[typing.Dict[str, typing.Any]]][List[Dict[str, Any]]], None]
Return list of views and their summaries.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.33.24 VertexCollection

class arango.collection.**VertexCollection**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor], graph: str, name: str*)

Vertex collection API wrapper.

Parameters

- **connection** – HTTP connection.
- **executor** – API executor.
- **graph** – Graph name.
- **name** – Vertex collection name.

graph

Return the graph name.

Returns Graph name.

Return type str

get(*vertex: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True*) → Union[Dict[str, Any], None, arango.job.AsyncJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]], arango.job.BatchJob[typing.Union[typing.Dict[str, typing.Any], NoneType]][Optional[Dict[str, Any]]]]

Return a vertex document.

Parameters

- **vertex** (*str | dict*) – Vertex document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns Vertex document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

insert(*vertex: Dict[str, Any], sync: Optional[bool] = None, silent: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Insert a new vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to insert. If it has "_key" or "_id" field, its value is used as key of the new vertex (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

update(*vertex: Dict[str, Any], check_rev: bool = True, keep_none: bool = True, sync: Optional[bool] = None, silent: bool = False, return_old: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Update a vertex document.

Parameters

- **vertex** (*dict*) – Partial or full vertex document with updated values. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **keep_none** (*bool | None*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace(*vertex: Dict[str, Any], check_rev: bool = True, sync: Optional[bool] = None, silent: bool = False, return_old: bool = False, return_new: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Replace a vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to replace the old one with. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **return_old** (*bool*) – Include body of the old document in the returned metadata. Ignored if parameter **silent** is set to True.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete(*vertex: Union[str, Dict[str, Any]], rev: Optional[str] = None, check_rev: bool = True, ignore_missing: bool = False, sync: Optional[bool] = None, return_old: bool = False*) → Union[bool, Dict[str, Any], arango.job.AsyncJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], arango.job.BatchJob[typing.Union[bool, typing.Dict[str, typing.Any]]][Union[bool, Dict[str, Any]]], None]

Delete a vertex document. All connected edges are also deleted.

Parameters

- **vertex** (*str | dict*) – Vertex document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | None*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool | None*) – Block until operation is synchronized to disk.
- **return_old** (*bool*) – Return body of the old document in the result.

Returns True if vertex was deleted successfully, False if vertex was not found and **ignore_missing** was set to True (does not apply in transactions). Old document is returned if **return_old** is set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

3.33.25 WAL

class arango.wal.**WAL**(*connection: Union[BasicConnection, JwtConnection, JwtSuperuserConnection], executor: Union[DefaultApiExecutor, AsyncApiExecutor, BatchApiExecutor, TransactionApiExecutor]*)

WAL (Write-Ahead Log) API wrapper.

properties() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return WAL properties.

Returns WAL properties.

Return type dict

Raises ***arango.exceptions.WALPropertiesError*** – If retrieval fails.

configure(*oversized_ops: Optional[bool] = None*, *log_size: Optional[int] = None*, *historic_logs: Optional[int] = None*, *reserve_logs: Optional[int] = None*, *throttle_wait: Optional[int] = None*, *throttle_limit: Optional[int] = None*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Configure WAL properties.

Parameters

- **oversized_ops** (*bool*) – If set to True, operations bigger than a single log file are allowed to be executed and stored.
- **log_size** (*int*) – Size of each write-ahead log file in bytes.
- **historic_logs** (*int*) – Max number of historic log files to keep.
- **reserve_logs** (*int*) – Max number of reserve log files to allocate.
- **throttle_wait** (*int*) – Wait time before aborting when write-throttled in milliseconds.
- **throttle_limit** (*int*) – Number of pending garbage collector operations that, when reached, activates write-throttling. Value of 0 means no throttling is triggered.

Returns New WAL properties.

Return type dict

Raises ***arango.exceptions.WALConfigureError*** – If operation fails.

transactions() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]

Return details on currently running WAL transactions.

Fields in the returned details are as follows:

```
"last_collected"    : ID of the last collected log file (at the
                      start of each running transaction) or None
                      if no transactions are running.

"last_sealed"       : ID of the last sealed log file (at the start
                      of each running transaction) or None if no
                      transactions are running.

"count"             : Number of currently running transactions.
```

Returns Details on currently running WAL transactions.

Return type dict

Raises *arango.exceptions.WALTransactionListError* – If retrieval fails.

flush(*sync: bool = True*, *garbage_collect: bool = True*) → Union[bool, arango.job.AsyncJob[bool][bool], arango.job.BatchJob[bool][bool], None]
Synchronize WAL to disk.

Parameters

- **sync** (*bool | None*) – Block until the synchronization is complete.
- **garbage_collect** (*bool*) – Block until flushed data is garbage collected.

Returns True if WAL was flushed successfully.

Return type bool

Raises *arango.exceptions.WALFlushError* – If flush operation fails.

tick_ranges() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the available ranges of tick values for all WAL files.

Returns Ranges of tick values.

Return type dict

Raises *arango.exceptions.WALTickRangesError* – If retrieval fails.

last_tick() → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Return the last available tick value (last successful operation).

Returns Last tick value in the WAL.

Return type dict

Raises *arango.exceptions.WALLastTickError* – If retrieval fails.

tail(*lower: Optional[str] = None*, *upper: Optional[str] = None*, *last_scanned: Optional[str] = None*, *all_databases: Optional[bool] = None*, *chunk_size: Optional[int] = None*, *syncer_id: Optional[int] = None*, *server_id: Optional[int] = None*, *client_info: Optional[str] = None*, *barrier_id: Optional[int] = None*, *deserialize: bool = False*) → Union[Dict[str, Any], arango.job.AsyncJob[typing.Dict[str, typing.Any]][Dict[str, Any]], arango.job.BatchJob[typing.Dict[str, typing.Any]][Dict[str, Any]], None]
Fetch recent WAL operations.

Parameters

- **lower** (*str | None*) – Exclusive lower bound tick value. On successive calls to this method you should set this to the value of "last_included" from previous call (unless the value was 0).
- **upper** (*str | None*) – Inclusive upper bound tick value for results.
- **last_scanned** (*str | None*) – On successive calls to this method you should set this to the value of "last_scanned" from previous call (or 0 on first try). This allows the rocksdb engine to break up large transactions over multiple responses.
- **all_databases** (*bool | None*) – Whether operations for all databases should be included. When set to False only the operations for the current database are included. The value True is only valid on "_system" database. The default is False.

- **chunk_size** (*int | None*) – Approximate maximum size of the returned result.
- **syncer_id** (*int | None*) – ID of the client used to tail results. The server will use this to keep operations until the client has fetched them. Must be a positive integer. Note this or **server_id** is required to have a chance at fetching reading all operations with the rocksdb storage engine.
- **server_id** (*int | None*) – ID of the client machine. If unset, the server will use this to keep operations until the client has fetched them. Must be a positive integer. Note this or **syncer_id** is required to have a chance at fetching reading all operations with the rocksdb storage engine.
- **client_info** (*str | None*) – Short description of the client, used for informative purposes only.
- **barrier_id** (*int | None*) – ID of barrier used to keep WAL entries around.
- **deserialize** (*bool*) – Deserialize the response content. Default is False.

Returns If **deserialize** is set to False, content is returned raw as a string. If **deserialize** is set to True, it is deserialized and returned as a list of dictionaries.

Return type dict

Raises ***arango.exceptions.WALTailError*** – If tail operation fails.

Python Module Index

a

Index

A

B

C

D

E

F

G

H

N

P

Q

R

S

T

U

V

W