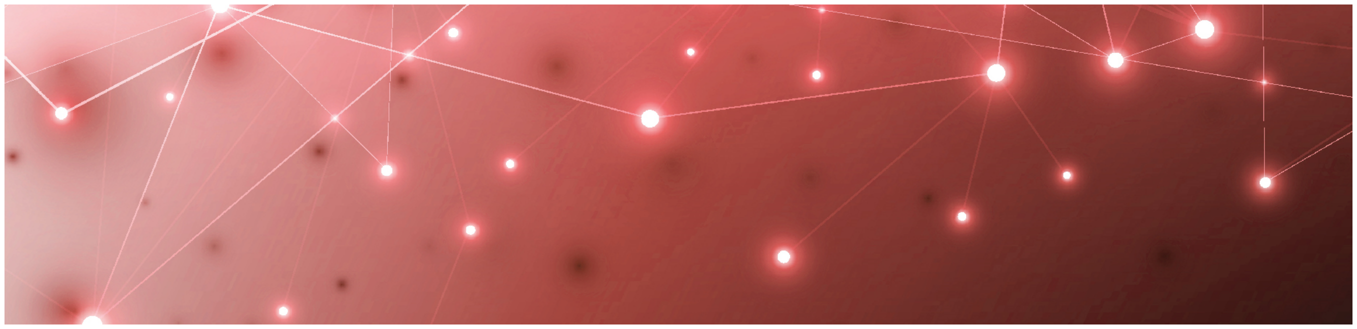


MARTELLO | ***Savision*** is a subsidiary of Martello Technologies

# Martello iQ

## REST API GUIDE

DOCUMENT DATE: MAY 13, 2021

Rest API Guide
Release 2.11 - May 13, 2021

# Contents

CHAPTER 1

Introduction ........ 4

Document Purpose and Intended Audience ........ 4

Revision History ........ 4

CHAPTER 2

About the REST API ........ 5

Basic Components ........ 5

Elasticsearch Node ........ 5

Elasticsearch Index ........ 5

Elasticsearch Type ........ 6

Elasticsearch Document ........ 6

Source ........ 6

Martello iQ Object Types ........ 6

Component ........ 7

Component State ........ 10

Component Relationship ........ 12

Hosting Relationship ........ 13

Containment Relationship ........ 14

Reference Relationship ........ 14

Alert ........ 15

Incident ........ 18

CHAPTER 3

Configuring an API Source ........ 21

CHAPTER 4

Using the REST API ........ 23

Create and Update a Document ........ 23

Retrieve a Document ........ 25

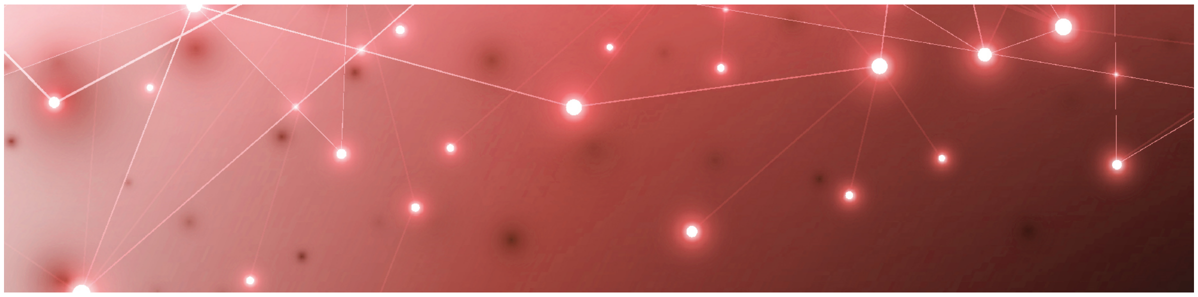
Delete a Document ........ 26

Batch Processing ........ 27

Search Documents ........ 28

List all Indexes ........ 29

Delete an Index ........ 29



CHAPTER 1

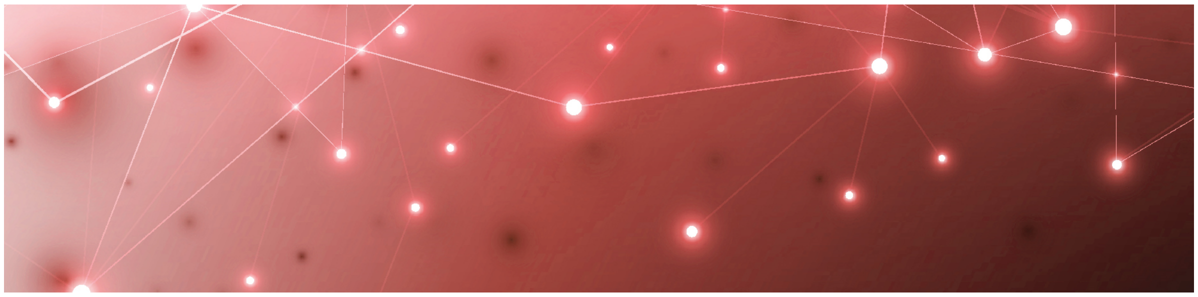
# Introduction

## Document Purpose and Intended Audience

This document contains information about configuring and using the Martello iQ Rest API.

## Revision History

| Document Date | Description |
|---|---|
| May 13, 2021 | Martello iQ Rest API Guide |



CHAPTER 2

# About the REST API

The Martello iQ REST API allows an external application to control the information stored in the Martello iQ Elasticsearch database. Use the information in the following sections to understand the core concepts of the Martello iQ REST API.

- "Basic Components" on page 5
- "Martello iQ Object Types" on page 6

## Basic Components

There are a few concepts that are core to Martello iQ and Elasticsearch. You must understand these concepts to successfully use the Martello iQ REST API.

### Elasticsearch Node

By default, Martello iQ installs a single Elasticsearch node co-located with the Martello iQ application in the same server. This node is responsible for storing the data that originates from different Sources, and provides indexing and search capabilities.

### Elasticsearch Index

An index is a collection of documents that have similar characteristics. In Martello iQ, there are specific indices for Components and States, Component Relationships, Alerts, and Incidents. An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it. The following index categories are defined in Martello iQ:

- **savisioniq_components_<SourceGuid>**—Each of these indices stores documents of type Component and Component State related to a specific Source.
- **savisioniq_component_relationships_<SourceGuid>**—Each of these indices stores documents of type Component Relationship related to a specific Source.
- **savisioniq_alerts_<SourceGuid>**—Each of these indices stores documents of type Alert related to a specific Source.

- **savisioniq_incidents_<SourceGuid>**—Each of these indices stores documents of type Incident related to a specific Source.

### Elasticsearch Type

Within an index, only one single Elasticsearch type is defined by Martello iQ. A type is defined for documents that have a set of common fields. The index category savisioniq_components_* contains the Elasticsearch type “esentity,” which may represent two different Martello iQ types: the parent type Component and the child type Component State. The following Elasticsearch types are defined in Martello iQ:

- **esentity**—This type is defined in the indices savisioniq_components_* and may represent the parent type Component and the child type Component State.
- **componentrelationship**—This type is defined in the indices savisioniq_ component_relationships_*.
- **alert**—This type is defined in the indices savisioniq_alerts_*.
- **incident**—This type is defined in the indices savisioniq_incidents_*.

### Elasticsearch Document

A document is a basic unit of information that can be indexed. For example, in Martello iQ you can have a document for a single Alert, another document for a single Component, and another for a single Incident. This document is expressed in JSON (JavaScript Object Notation), which is a ubiquitous internet data interchange format. Within an index, you can store as many documents as you want.

Each document in an index has an _id field. The _id field is used to uniquely identify a document within that index. Elasticsearch searches on the _id, rather than on the text of a document, to locate information quickly. For example, when you perform a query or use the GET API operation, Elasticsearch uses the _id to look up documents.

### Source

In Martello iQ a Source (also called Integration in the interface) represents a specific monitoring tool or ITSM system—such as SCOM or ServiceNow—that provides the data to be stored into Elasticsearch. Each Source is uniquely identified by a GUID.

## Martello iQ Object Types

The Martello iQ REST API has five different types of objects:

- "Component" on page 7
- "Component State" on page 10
- "Component Relationship" on page 12
- "Alert" on page 15
- "Incident" on page 18

You must provide a valid object of a specific type to correctly create and update a document in Elasticsearch using the Martello iQ REST API.

## Component

A Component is defined as follows:

```
esentity

{

     joinKey (join)
     key (text)
     sourceId (text)
     sourceName (text)
     sourceType (text)
     source (object)
     name (text)
     typeEnum (integer)
     host (text)
     path (text)
     url (text)
     iPAddress (text)
     fqdn (text)
     geoLocation (geo_point)

}
```

For additional information about the Elasticsearch datatypes see:

https://www.elastic.co/guide/en/elasticsearch/reference/6.8/mapping-types.html

The following table lists the fields in the Component document:

| Field | Description |
|---|---|
| joinKey | This is a special field that creates a parent/child relationship between one Component (the parent) and the associated Component States (the children). For a Component it must always be set to “parent." |
| key | The unique identifier of the Component document. A key is composed of two parts that are separated by a pipe character \|. The first part is the ID of the Source of the Component; the second part is a string the uniquely identifies that Component among all the Components of the Source. |

| Field | Description |
|---|---|
| sourceId | The unique identifier of the Source related to the Component. The sourceId is the string representation of a globally unique identifier (GUID). |
| sourceName | The name of the Source related to the Component. |
| sourceType | The type of the Source related to the Component, for example “SCOM” or “ServiceNow." For a Martello iQ API Source the type must always be “VirtualConnector." |
| source | An object that contains the raw properties of the Component specific to the Source. |
| name | The name of the Component object. |
| typeEnum | An integer that represents the type of the Component. The possible Component types are:<br>• 1: Object<br>• 2: Group<br>• 3: Service<br>• 4: Computer<br>• 5: Database<br>• 6: Website<br>• 7: Virtual Machine |
| host | The name of the Component that hosts the Component. |
| path | The path that identifies Source related to the Component. |
| url | The URL that locates the Component in the monitoring/ITSM system. You can use this URL to navigate to the page of the monitoring/ITSM system that contains specific information about this Component. |
| iPAddress | The IP address(es) associated with the Component. |
| fqdn | The fully qualified domain name associated with the Component. |
| geoLocation | The geolocation of the Component.<br>For information about how to specify a geoLocation field, see https://www.elastic.co/guide/en/elasticsearch/reference/6.8/geo-point.html |

An example of a SCOM Component in JSON format is the following:

```
{
     "joinKey": "parent",
     "name": "TestDB1",
     "typeEnum": 5,
     "path": "BRSRV2012R2-2.savisionlab.Savision.int;MSSQLSERVER",
     "url": "/ManagedEntity/ManagedEntity/9a84ffd3-5b42-85e7-c242-
     9264cd6a62e9",
     "fqdn": "fqdn": "BRSRV2012R2-2.savisionlab.Savision.int",
     "key": "12b3b4b4-ec8b-4d03-8b64-19f90c47c1ab|9a84ffd3-5b42-85e7-
     c242-9264cd6a62e9",
     "sourceId": "12b3b4b4-ec8b-4d03-8b64-19f90c47c1ab",
     "sourceName": "Unity iQ SCOM 01",
     "sourceType": "SCOM",
     "source": {
           "scom": {
                 "Object Display Name": "TestDB1",
                 "FullName":
     "Microsoft.SQLServer.2014.Database:BRSRV2012R2-
     2.savisionlab.Savision.int;MSSQLSERVER;TestDB1",
                 "Id": "9a84ffd3-5b42-85e7-c242-9264cd6a62e9",
                 "ManagementGroupName": "savisonUnityiQ",
                 "Path": "BRSRV2012R2-
     2.savisionlab.Savision.int;MSSQLSERVER",
                 "TimeAdded": "2017-02-07T18:04:24.4991017",
                 "Database Name": "TestDB1",
                 "Recovery Model": "",
                 "Database Autogrow Set": "",
                 "Log Autogrow Set": "",
                 "Updateability": "",
                 "User Access": "",
                 "Collation": "",
                 "Owner": "",
                 "Resource Pool": "",
                 "Object Status":
     "System.ConfigItem.ObjectStatusEnum.Active",
                 "Asset Status": "",
                 "Notes": "",
                 "Display Name": "TestDB1",
                 "Instance Name": "MSSQLSERVER",
                 "Principal Name": "BRSRV2012R2-
     2.savisionlab.Savision.int"
```

```
                }
        }
}
```

In order to create, update, delete, and retrieve a Component or a batch of Components, you need to specify the routing parameter in the request. The routing parameter must equal the `key` property of the component. See the section "Create and Update a Document" on page 23 for an example of a request.

## Component State

A Component State is defined as follows:

```
esentity
{
     joinKey (join)
     sourceId (text)
     sourceName (text)
     sourceType (text)
     source (object)
     componentKey (text)
     stateIndex (integer)
     state (text)
     timestamp (date)
     lastSyncTime (date)
     isCurrent (boolean)
}
```

The Elasticsearch _id of the esentity document must be specified only for current Component States.

In this case the _id has the following format:

`"<componentKey>|STATE"`

where `<componentKey>` is the componentKey field of the Component State.

The following table lists the fields in the Component State document:

| Field | Description |
|---|---|
| joinKey | This is a special field that creates a parent/child relationship between one Component (the parent) and the associated Component States (the children).<br><br>For a Component State it must be set to the following object:<br><br>`{`<br>`"name": "esentity",`<br>`"parent": "<componentKey>"`<br>`}`<br><br>where `<componentKey>` is the `componentKey` field of the Component State. |
| sourceId | The unique identifier of the Source related to the Component States. The sourceId is the string representation of a globally unique identifier (GUID). |
| sourceName | The name of the Source related to the Component States. |
| sourceType | The type of the Source related to the Component States, for example “SCOM” or “AWS." For a Martello iQ API Source the type must always be “VirtualConnector." |
| source | An object that contains the raw properties of the Component States specific to the Source. |
| componentKey | The unique identifier of the Component associated with the Component State. |
| stateIndex | An integer that represents the health state of the Component State. |

| Field | Description |
|---|---|
| state | The health state of the Component State. The Health States and their indices are:<br>• 0. Unknown<br>• 1. Unreachable<br>• 2. Not Monitored<br>• 3. In Maintenance Mode<br>• 4. Healthy<br>• 5. Warning<br>• 6. Critical |
| timestamp | The time the Component changed its state. |
| lastSyncTime | The last time the current Component State was updated. If null, the Component State represents a historical state. |
| isCurrent | A boolean that indicates whether the Component State represents the current state of the Component or a historical state. If lastSyncTime is null, isCurrent must be set to false; otherwise it must be set to true. |

In order to create, update, delete, and retrieve a Component State or a batch of Component States, you need to specify the routing parameter in the request. The routing parameter must equal the `componentKey` field of the Component State.

## Component Relationship

A Component Relationship is defined as follows:

```
componentrelationship
{
     key (text)
     sourceId (text)
     sourceName (text)
     sourceType (text)
     source (object)
     name (text)
     sourceComponent (text)
     destinationComponent (text)
     typeEnum (integer)
}
```

The following table lists the fields in the Component Relationships document:

| Field | Description |
|---|---|
| key | The unique identifier of the Component Relationship document. A key is composed of two parts that are separated by a pipe character \|. The first part is the ID of the Source of the Component Relationship; the second part is a string that uniquely identifies the Relationship among all the Relationships of the Source. |
| sourceId | The unique identifier of the Source related to the Component Relationship. The sourceId is the string representation of a globally unique identifier (GUID). |
| sourceName | The name of the Source related to the Component Relationship. |
| sourceType | The type of the Source related to the Component Relationship, for example “SCOM” or “ServiceNow." For a Martello iQ API Source, the type must always be “VirtualConnector." |
| source | An object that contains the raw properties of the Component Relationship specific to the Source. |
| name | The name of the Component Relationship object. |
| sourceComponent | The key (unique identifier) of the Component that represents the “source” in the relationship. |
| destinationComponent | The key (unique identifier) of the Component that represents the “destination” in the relationship. |
| typeEnum | An integer that represents the type of the Component Relationship. The Component Relationship types and their values are:<br>• 1: Hosting<br>• 2: Containment<br>• 3: Reference |

### Hosting Relationship

The most restrictive relationship between Components is the hosting relationship. When one Component is hosted by another, that Component relies on its hosting parent for its very existence. If the hosting parent is removed, the hosted child will also be removed. For example, a logical disk cannot exist without the computer that it is installed on. A hosted Component can have only one hosting parent, but one parent can host multiple children. For example, a particular disk can be installed on only a single computer, but one computer can have several disks installed.

### Containment Relationship

The containment relationship type is less restrictive than the hosting relationship. It declares that one Component is related to another Component, although one is not required for the other. Unlike a hosting relationship, a containment relationship is many-to-many. This means that one Component can contain multiple Components, and a single Component can be contained by multiple other Components. For example, one group can contain multiple components, and a single component can be a member of multiple groups.

### Reference Relationship

The reference relationship is the most general relationship type. A reference relationship is used when the source and destination components are not dependent on one another; for example, a database can reference another database that it is replicating. One database is not dependent on the other, and the Components exist separately.

An example of a SCOM Component Relationship in JSON format is the following:

```
{
    "name": "BRSRV2012R2-2.savisionlab.Savision.int\\MSSQLSERVER Hosts
    BRSRV2012R2-2.savisionlab.Savision.int\\MSSQLSERVER\\TestDB1",
    "sourceComponent": "12b3b4b4-ec8b-4d03-8b64-19f90c47c1ab|da967a6a-
    0ac8-57fc-287e-2b21f818ca13",
    "destinationComponent": "12b3b4b4-ec8b-4d03-8b64-
    19f90c47c1ab|9a84ffd3-5b42-85e7-c242-9264cd6a62e9",
    "typeEnum": 1,
    "type": "Hosting",
    "key": "12b3b4b4-ec8b-4d03-8b64-19f90c47c1ab|d6a9160b-785c-388f-
    d09d-32c5a4861ea4",
    "sourceId": "12b3b4b4-ec8b-4d03-8b64-19f90c47c1ab",
    "sourceName": "Unity iQ SCOM 01",
    "sourceType": "SCOM",
    "source": {
          "scom": {
                "Object Display Name": "MSSQLSERVER - TestDB1",
                "TypeName": "MSSQL 2014: SQL Server 2014 Database
    Engine Hosts SQL Database",
                "TypeDescription": null,
                "SourceDisplayName": "MSSQLSERVER",
                "SourceFullName":
    "Microsoft.SQLServer.2014.DBEngine:BRSRV2012R2-
    2.savisionlab.Savision.int;MSSQLSERVER",
                "SourceId": "da967a6a-0ac8-57fc-287e-2b21f818ca13",
                "SourcePath": "BRSRV2012R2-
    2.savisionlab.Savision.int",
                "TargetDisplayName": "TestDB1",
                "TargetFullName":
    "Microsoft.SQLServer.2014.Database:BRSRV2012R2-
    2.savisionlab.Savision.int;MSSQLSERVER;TestDB1",
                "TargetId": "9a84ffd3-5b42-85e7-c242-9264cd6a62e9",
                "TargetPath": "BRSRV2012R2-
    2.savisionlab.Savision.int;MSSQLSERVER",
                "ManagementGroupName": "savisonUnityiQ",
                "LastModified": "2017-02-07T18:04:24.673"
          }
    }
}
```

## Alert

An Alert is defined as follows:

```
alert
{
     key (text)
     sourceId (text)
     sourceName (text)
     sourceType (text)
     source (object)
     name (text)
     componentKey (text)
     target (text)
     severityIndex (integer)
     severity (text)
     message (text)
     isActive (boolean)
     isAcknowledged (boolean)
     resolutionState (text)
     created (date)
     lastUpdated (date)
     assignee (text)
     url (text)
}
```

The Elasticsearch _id of the alert document must be set equal to the `key` field of the Alert.

The following table lists the fields in the Alert document:

| Field | Description |
|---|---|
| joinKey | This is a special field that creates a parent/child relationship between one Component (the parent) and the associated Component States (the children). For a Component it must always be set to “parent." |
| key | The unique identifier of the Component document. A key is composed of two parts that are separated by a pipe character \|. The first part is the ID of the Source of the Component; the second part is a string the uniquely identifies that Component among all the Components of the Source. |

| Field | Description |
|---|---|
| sourceId | The unique identifier of the Source related to the Component. The sourceId is the string representation of a globally unique identifier (GUID). |
| sourceName | The name of the Source related to the Component. |
| sourceType | The type of the Source related to the Component, for example “SCOM” or “ServiceNow." For a Martello iQ API Source, the type must always be “VirtualConnector." |
| source | An object that contains the raw properties of the Alert specific to the Source. |
| name | The name of the Alert object. |
| componentKey | The unique identifier of the Component associated with the Alert. |
| target | The name of the Component associated with the Alert. |
| severityIndex | An integer that represents the severity of the Alert. |
| severity | The severity of the Alert. The severity levels are:<br>• 1. Information<br>• 2. Warning<br>• 3. Error |
| message | The detailed description of the Alert. |
| isActive | A boolean that indicates whether the Alert is active or resolved/closed. |
| isAcknowledged | A boolean that indicates whether the Alert is acknowledged. |
| resolutionState | The resolution state of the Alert. |
| created | The time the Alert was raised. |
| lastUpdated | The last time the Alert was modified. |
| assignee | The owner of the Alert. |

| Field | Description |
|---|---|
| url | The URL that locates the Alert in the monitoring system. You can use this URL to navigate to the page of the monitoring system that contains specific information about this Alert. |

## Incident

An Incident is defined as follows:

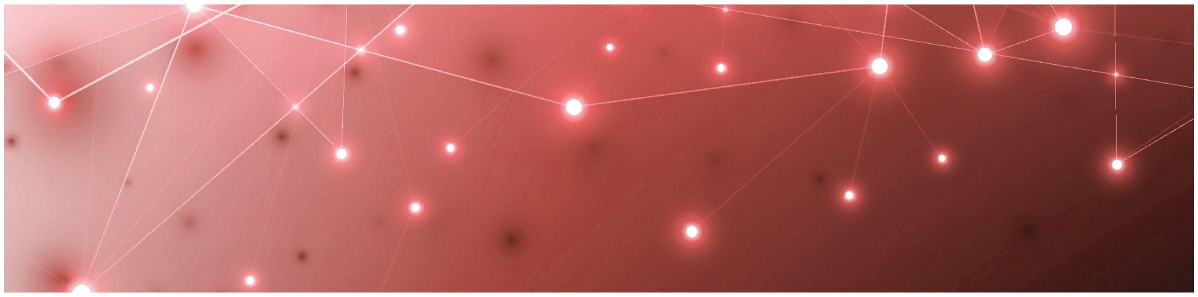
```
incident
{
     key (text)
     sourceId (text)
     sourceName (text)
     sourceType (text)
     source (object)
     name (text)
     componentKey (text)
     target (text)
     impact (text)
     urgency (text)
     priority (text)
     message (text)
     description (text)
     isActive (boolean)
     created (date)
     lastUpdated (date)
     state (text)
     assignedTo (text)
     url (text)
}
```

The Elasticsearch _id of the incident document must be set equal to the `key` field of the Incident.

The following table lists the fields in the Incident document:

| Field | Description |
|---|---|
| key | The unique identifier of the Incident document. A key is composed of two parts that are separated by a pipe character \|. The first part is the ID of the Source of the Incident; the second part is a string the uniquely identifies that Incident among all the Incidents of the Source. |
| sourceId | The unique identifier of the Source related to the Incident. The sourceId is the string representation of a globally unique identifier (GUID). |
| sourceName | The name of the Source related to the Incident. |
| sourceType | The type of the Source related to the Incident, for example “ServiceNow." For a Martello iQ API Source, the type must always be “VirtualConnector." |
| source | An object that contains the raw properties of the Incident specific to the Source. |
| name | The name of the Incident object. |
| componentKey | The unique identifier of the Component associated with the Incident. |
| target | The name of the Component associated with the Incident. |
| impact | The impact of the Incident. A measure of the effect of the Incident. |
| urgency | The urgency of the Incident. A measure of how quickly a resolution of the Incident is required. |
| priority | The priority of the Incident, derived from urgency and impact. |
| message | The short description of the Incident. |
| isActive | A boolean that indicates whether the Incident is active or resolved/closed. |
| created | The time the Incident was created. |
| lastUpdated | The last time the Incident was modified. |
| state | The current state of the Incident. |

| Field | Description |
|---|---|
| assignedTo | The owner of the Incident. |
| url | The URL that locates the Incident in the ITSM system. You can use this URL to navigate to the page of the ITSM system that contains specific information about this Incident. |



CHAPTER 3

# Configuring an API Source

Before starting to use the Martello iQ REST API, you need to configure an Open API source in Martello iQ. Use the following procedure to configure the source.

1. Connect to Martello iQ using a browser.
2. From the main menu, select **Settings**.
   The Integrations page displays.
3. Click the **Add** button at the bottom of the page.
4. Select **Martello iQ API**.
5. Enter the following information and click **Save**:

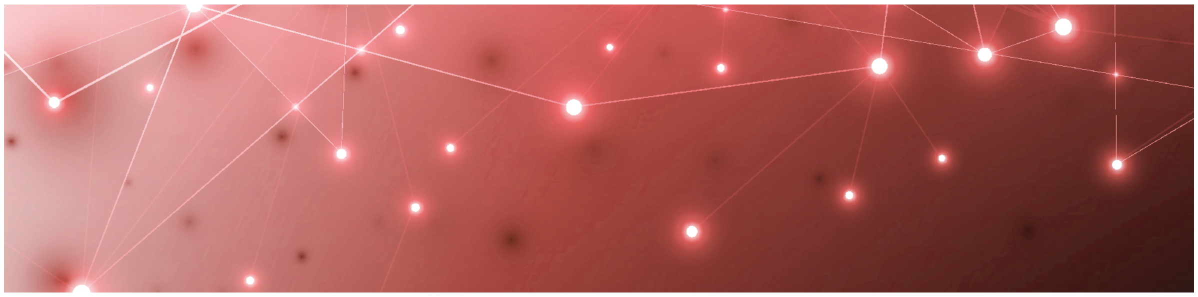
| Property | Description |
|---|---|
| Agent | Select a server to communicate with the source system. This can be the Martello iQ web server or a machine that has a Martello iQ Remote agent installed on it. |
| Name | Provide a name for the integration; this name displays on the Martello iQ interface. |
| Discovery Interval | How often the objects are loaded from the integrated system. The default is 3600 seconds. |
| Operation Interval | How often health states, alerts, and/or incidents are collected. The default is 120 seconds. |

6. To view and copy the source GUID, click the 🔑 button.

**Note:**
It is important to know the GUID because it is used to define the indices of Elasticsearch. The source GUID is reported in the property `sourceGuid` and it is used to populate the property `sourceId` of any Document created with the Martello iQ REST

API. The Connector Type is always VirtualConnector and it is used to populate the property `sourceType` of the created Document.

7. Click the **Roles** tab and select a role.
   A new page displays.
8. Click a role and select **Integrations**.
9. Click the **Add** button.
10. Select an integration from the list and click **Add**.
11. Optional. If you want users in this role to have read-only access to the integration, select the **Read-only** box.



CHAPTER 4

# Using the REST API

In order to communicate with the Elasticsearch node using the Martello iQ REST API, you need to have HTTP access to the Martello iQ server address and port 9200.

By default, the Elasticsearch node binds to loopback addresses only, i.e. 127.0.0.1 and [::1]. If you want to have access from other servers, the Elasticsearch node will need to bind to a non-loopback address. In the Elasticsearch configuration file “elasticsearch.yml” modify the setting “network.host” to make the node bind to a different hostname or IP address. For more details, please consult the Elasticsearch documentation.

The example below shows the pattern of how to access data in Elasticsearch:

```
<REST Verb> <Martello iQ Server>:9200/<Index>/<Type>/<_id>
```

The REST verbs used to interact with Elasticsearch are GET (Retrieve/List), PUT (Create/Update), DELETE (Delete) and POST (Batch Processing).

For information about the <Index>, <Type>, and <_id> variables, see "Basic Components" on page 5

You can use command-line tools like curl , PowerShell’s Invoke-RestMethod or visual tools like Postman to perform REST requests.

To illustrate the Martello iQ REST API, we will make use of the curl syntax in the following examples. For simplicity the Unix syntax is adopted; if you use curl in a Microsoft Windows command prompt, you should replace all the single quotes ( ' ) with double quotes ( " ) and any double quote ( " ) inside single quotes with two double quotes ( "" ).

## Create and Update a Document

To create or update an existing document, use the Index API to specify the index, the type and the _id of the document.

The basic syntax for indexing a document is:

```
PUT <Martello iQ Server>:9200/<Index>/<Type>/<_id>
```

You must provide the object to persist in JSON format in the body of the request. The _id of the document must be equal to the key property of the object to persist.

For example, to index a simple Component document into the savisioniq_components_075261f3-e421-41d7-b71b-8320dccbf194 index, using the curl tool:

```
curl -XPUT 'localhost:9200/savisioniq_components_075261f3-e421-41d7-
b71b-8320dccbf194/esentity/075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001?routing=075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001' -H 'Content-Type: application/json' -d'

{

     "joinKey": "parent",

     "name": "Testing Server 1",

     "typeEnum": 4,

     "key": "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000001",

     "sourceId": "075261f3-e421-41d7-b71b-8320dccbf194",

     "sourceName": "Samanage 01",

     "sourceType": "VirtualConnector",

     "source": {

           "api": {

                 "DisplayName": "Testing Server 1",

                 "Id": "SRV0000001",

                 "TimeAdded": "2017-02-07T18:04:24.4991017",

           }

     }

}

'
```

Note how the Document _id matches the Component key. In addition, you need to specify the routing parameter in the request, and the routing parameter must equal the **key** property of the component.

And the response:

```
{
     "_index" : "savisioniq_components_075261f3-e421-41d7-b71b-
     8320dccbf194",
     "_type" : "esentity",
     "_id" : "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000001",
     "_version" : 1,
     "result" : "created",
     "_shards" : {
     "total" : 1,
     "successful" : 1,
     "failed" : 0
     },
     "created" : true
}
```

From the above, we can see that a new document of type esentity was successfully created inside the specified index. The document also has an internal id of "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000001" which we specified at index time.

It is important to note that Elasticsearch does not require you to explicitly create an index first before you can index documents into it. In the previous example, Elasticsearch will automatically create the index "savisioniq_components_075261f3-e421-41d7-b71b-8320dccbf194" if it didn't already exist beforehand.

To update an existing document you can execute the above command again with a different (or same) JSON object; in fact Elasticsearch will replace (i.e. re-index) a new document on top of the existing one with the _id of "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000001".

If, on the other hand, you use a different _id, a new document will be indexed and the existing documents in the index remain untouched.

## Retrieve a Document

To retrieve an existing document, use the Get API operation and specify the index, the type and the _id of the document.

The basic syntax for retrieving a document is:

```
GET <Martello iQ Server>:9200/<Index>/<Type>/<_id>
```

The following example shows how use the curl tool to retrieve the Component document that was previously indexed:

```
curl -XGET 'localhost:9200/savisioniq_components_075261f3-e421-41d7-
b71b-8320dccbf194/esentity/075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001?routing=075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001'
```

You need to specify the routing parameter in the request, and the routing parameter must equal the **key** property of the component.

Here is the response:

```
{
     "_index" : "savisioniq_components_075261f3-e421-41d7-b71b-
     8320dccbf194",
     "_type" : "esentity",
     "_id" : "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000001",
     "_version" : 1,
     "found" : true,
     "_source" : {
                 "joinKey": "parent",
                 "name": "Testing Server 1",
                 "typeEnum": 4,
                 "key": "075261f3-e421-41d7-b71b-
     8320dccbf194|SRV0000001",
                 "sourceId": "075261f3-e421-41d7-b71b-8320dccbf194",
                 "sourceName": "Samanage 01",
                 "sourceType": "VirtualConnector",
                 "source": {
                    "api": {
                       "DisplayName": "Testing Server 1",
                       "Id": "SRV0000001",
                       "TimeAdded": "2017-02-07T18:04:24.4991017",
                    }
           }
     }
}
```

The response field found, reports if the requested document was found, while the field _source returns the full JSON document that was previously indexed.

## Delete a Document

To delete an existing document, use the Delete API operation and specify the index, the type and the _id of the document.

The basic syntax for deleting a document is:

```
DELETE <Martello iQ Server>:9200/<Index>/<Type>/<_id>
```

The following example shows how to use the curl tool to delete the Component document that was previously indexed:

```
curl -XDELETE 'localhost:9200/savisioniq_components_075261f3-e421-
41d7-b71b-8320dccbf194/esentity/075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001?routing=075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000001'
```

You need to specify the routing parameter in the request, and the routing parameter must equal the **key** property of the component.

See the _delete_by_query API to delete all documents that match a specific query. It is more efficient to delete a whole index instead of deleting all documents with the Delete By Query API.

The response field reports if the requested document was found, while the field _ source returns the full JSON document that was previously indexed.

## Batch Processing

In addition to being able to index and delete individual documents, Elasticsearch also provides the ability to perform these operations in batches using the bulk API. This functionality is important in that it provides a very efficient mechanism to do multiple operations as fast as possible with as few network roundtrips as possible.

The basic syntax for indexing a document is:

```
POST <Martello iQ Server>:9200/<Index>/<Type>/_bulk
```

You must provide the following newline delimited JSON (NDJSON) structure in the body of the request:

```
action_and_meta_data\n

optional_source\n

action_and_meta_data\n

optional_source\n

…

action_and_meta_data\n

optional_source\n
```

For example, to index two Component documents in one bulk operation:

```
curl -XPOST 'localhost:9200/savisioniq_components_075261f3-e421-41d7-
b71b-8320dccbf194/esentity/_bulk' -H 'Content-Type: application/json'
-d'

{"index":{"_id":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000002","routing":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000002"}}

{"joinKey":"parent","name":"Testing Server 2","typeEnum": 4,"key":
"075261f3-e421-41d7-b71b-8320dccbf194|SRV0000002", … }

{"index":{"_id":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000003","routing":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000003"}}

{"joinKey":"parent","name":"Testing Server 3","typeEnum": 4,"key":
"075261f3-e421-41d7-b71b-8320dccbf194|SRV0000003", … }

'
```

You need to specify the routing parameter in the request, and the routing parameter must equal the **key** property of the component.

This example re-indexes the first document and then deletes the second document in one bulk operation:

```
curl -XPOST 'localhost:9200/ savisioniq_components_075261f3-e421-41d7-
b71b-8320dccbf194/esentity/_bulk ' -H 'Content-Type: application/json'
-d'

{"index":{"_id":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000002"","routing":"075261f3-e421-41d7-b71b-
8320dccbf194|SRV0000002"}}

{"joinKey":"parent","name": "Renamed Testing Server 2","typeEnum":
4,"key": "075261f3-e421-41d7-b71b-8320dccbf194|SRV0000002", … }

{"delete":{"_id":"075261f3-e421-41d7-b71b-8320dccbf194|SRV0000003"}}

'
```

The example above shows that for deletions, there is no corresponding source document because deletions require the _id of the document only.

The Bulk API does not fail due to failures in one of the actions. If a single action fails for whatever reason, it will continue to process the remainder of the actions after it. When the bulk API returns, it will provide a status for each action (in the same order it was sent in) so that you can check if a specific action failed or not.

## Search Documents

To search the documents persisted into Elasticsearch, you use the Search API. Refer to the Elasticsearch documentation for details.

## List all Indexes

To get information about all the indexes, you use the cat indices API. The basic syntax is:

```
GET <Martello iQ Server>:9200/_cat/indices
```

Using the curl tool, the basic syntax is:

```
curl -XGET 'localhost:9200/_cat/indices'
```

## Delete an Index

To delete an existing index, you use the Delete Index API, specifying the index.

The basic syntax for deleting an index is:

```
DELETE <Martello iQ Server>:9200/<Index>
```

The following example shows how to use the curl tool to delete an index that was previously created:

```
curl -XDELETE 'localhost:9200/savisioniq_components_075261f3-e421-
41d7-b71b-8320dccbf194'
```