Resource Reference Guide

Introduction

The Vantiq REST and WebSocket APIs provide programmatic access to the resources managed by the Vantiq server. This includes both system defined resources provided by Vantiq and custom resources created by Vantiq developers as part of their “event driven applications” (EDA). After presenting some general information about resources, this document focuses on the Vantiq system resources. For information about how to define and access custom resources, refer to the API Reference Guide.

Resource Definition

All resources have a “type” (referred to as a “resource type”) which describes the structure of the resource’s instances. The type defines the properties available and characteristics for each property such as its VAIL type, whether or not it is required, any default value, and a description (see the types resource for more details).

Each resource instance is uniquely identified by its “resource id”. For system resources, the definition of the resource id is part of the resource type definition. For custom resources it is always the system defined `_id` property.

Resource Security Model

As described in the API Reference Guide, all resource requests must be made by an authenticated user identity. When this identity is established, the system creates a security context which includes:

- The namespace to which the user is bound.
- The profiles available to the user.

Users have potential access to resource instances in two namespaces, the one to which they are bound and the system namespace. The precise details of what access is permitted for each specific resource is governed by the user’s assigned profiles. Each user has exactly 2 profiles – one which governs access to their local namespace and one which governs access to the system namespace. For each of these, the system creates 2 default profiles: user and admin. The administrator of a namespace may create additional, non-default profiles for their namespace and assign these to users in that namespace as desired (bearing in mind that each user must have exactly one such profile).

In the description of each resource we will outline the access levels granted for each resource by the default user and admin profiles. The description of the Types resource includes a discussion of how to describe the default access levels for a user-defined type and how to create custom access levels for use when designing custom profiles.

Default Access Levels

The default access levels that are available when defining any resource type are:

- **ars_denied** - grant no access to the resource.
- **ars_readOnly** - grant read access to all available resource instances.
- **ars_readWrite** - grant read-write access to all available resource instances.
- **ars_readByOwner** - grant read access to resource instances which have been created by the requesting user.
- **ars_writeByOwner** - grant read-write access to resource instances which have been created by the requesting user.

Public Resources

To access any resource, regardless of access level, the user is required to have some level of authorization. That authorization is typically communicated by an access token which identifies the user, and by extension their associated authorizations in a namespace. To expose access to an instance of any system resource without any authorization at all, set the **ars_public** system property of the instance to `true`.

For more information on how to interact with public resources via the REST API, please refer to the API Reference Guide.

Resource Relationship Model

The resource relationship model describes the semantic relationships that exist between the resources that comprise a complete Vantiq Application. This relationship data can be useful when analyzing or visualizing the elements of a Vantiq Application. For example, Modelo uses the relationship model to construct the Resource Graph.

Vantiq will try to infer the possible relationships between resources. However, there are cases in which inference may fall short such as when resources are referred to using hard-coded strings or in Client Buidler javascript code. In these cases, users may create explicit relationships between resources as well.

For more information see the API Reference Guide.

The following are the possible relationships between two resources:

- **dependency**: One resource depends on another resource’s existence. *Example*: A Procedure which inserts into a Type has a dependency on that Type.

- **executes**: A resource that executes a Procedure has an *executes* relationship to that Procedure.
- **publishes**: A resource that publishes to a Topic or Source has a *publishes* relationship to that resource
- **subscribes_to**: A resource that is subscribed to events from another resource. *Example*: A rule triggered by publishes to a Topic has a *subscribed_to* relationship to that Topic.
- **implements**: The relationship between a generated Service and the App or Collaboration Type it implements
- **contains**: The relationship between a meta-resource and the resources it contains. *Example*: A project has a *contains* relationship with all the resources in the Project. A Service has a *contains* relationship with all of the Procedures in that Service.
- **triggers**: The relationship between an App and a Collaboration Type that is started by that App
- **tests**: The relationship between a Test or Test Suite and the resource (Procedure, Rule, Project) that it tests.

The *Relationships* sub-section for each Vantiq resource describes the relationships that may be inferred for that resource (if any).

Vantiq Resources

The following resources are part of the core resource model and available for use by all Vantiq applications:

- audits: objects created automatically to record important events within the system
- assemblyconfigs: contains the configuration for a particular assembly
- captures: contains all of the recorded events from an event capture
- configurations: defines sets of rules which share the same activation state
- debugconfigs: configures the logging and tracing levels for groups of rules and procedures
- delegatedrequests: defines a resource operation which can be executed through use of a one-time “code”. This allows the execution to be deferred in time and also delegated from one user to another.
- documents: static content generally loaded from a file
- eventgenerators: defines a resource which can generate any type, topic, or source event
- groups: defines a collection of users with access to a set of resources.
- images: static content generally loaded from file. Restricted to image types.
- k8sclusters: information about Kubernetes clusters with managed resources
- k8sinstallations: managed installations in Kubernetes clusters
- k8sworkitems: transient work to be performed in Kubernetes clusters
- llms: represents a Large Language Model (LLM).
- logs: log messages written using the built in `log` procedure.
- namespaces: defines a namespace allocated to an organization
- nodes: identifies remote Vantiq federated installation(s)
- organizations: a tenant who has been provisioned to manage a collection of users and namespaces.
- procedures: a procedure available for use in rules and other procedures as documented in the Rule and Procedure Reference Guide
- profiles: defines capabilities assigned to a user
- projects: defines a collection of Vantiq resources that is a deployable unit
- rules: defines user-created rules for processing incoming data
- secrets: passwords, tokens, certificates, or any other secure piece of text that can be used to configure sources without exposing the secret value in the configuration
- semanticindexes: some content which has been indexed and made available for Semantic Search.
- ArsSemanticIndexEntry: the entries available for Semantic Searches against Semantic Indexes.
- serviceconnectors: the implementation of a service via an external process.
- services: encapsulate behavior associated with a specific functional aspect of an application, including any interactions with the application data model related to that behavior.
- scheduledevents: events that have been scheduled to be delivered at some point in the future. The event may be delivered once or on a periodic basis.
- sources: external sources of data, e.g. MQTT, AMQP, RESTful
- storagemanagers: managers for external data stores, enable “plug-in” of remotely accessible databases, e.g. InfluxDB Cloud or MongoDB Atlas
- tempblobs: short-term, memory-resident larger objects
- tensorflowmodels: TensorFlow models used for neural net processing. (TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.)
- tests: A test for a Rule, Procedure, or Project defined by a series of input events and expected asynchronous output events
- testsuites: A series of tests for a particular resource
- testreports: The result of a test run containing any errors that may have occurred during the test
- topics: user-created identifiers that may be published to drive rules and trigger notifications
- trackingRegions: user-created areas used in motion tracking.
- types: defines the objects the Vantiq System operates on. Types represent traditional objects stored in a database.
- users: authorized users of Vantiq
- videos: static content generally loaded from file. Restricted to video types.

The section for each resource type describes the type’s structure and the available operations. It also describes the access privileges granted by the built-in admin and user profiles. By default, admin users will have full access to all resource instances in their namespace and users will have read-only access to all resource instances in their namespace. There are exceptions, however, which are described below.

Note that all examples in this document illustrate the use of the REST API to manage the resources. Refer to the WebSockets section of the API Reference Guide for general information on managing resources using the WebSocket API.

The Vantiq server may have an environment specific domain name. In this document the default development server, https://dev.vantiq.com, is the server used in statement syntax blocks and examples.

Resource Events

Some resources generate events when specific operations are performed. These events can be referenced in the WHEN statement of a rule or event stream, allowing them to trigger user-defined processing whenever the event occurs. Event bindings use the "path" of the event which is always of the form: `/<resource>/<resourceId>/<op>` (for example `/namespaces/myNamespace/authorized`). The operations that trigger events and the data associated with them is described in the *Events* section of each resource description.

Assembly Configs

Assembly Configurations contain the consumer supplied configurations for an Assembly.

Assembly Configurations have the following properties:

- **name**: the unique identifier for the assembly configuration. This will default to the name of the assembly being configured
- **assembly**: the name of the assembly being configured
- **configuration**: an object containing the configuration properties. That is, key-value pairs where the keys are the names of the configuration properties of the Assembly and the values are the consumer supplied values for each property. See the Assembly Guide for more details.

Inserting or updating the *assemblyconfig* will automatically apply the configurations to the resources in the Assembly according to the configuration mappings. Deleting the *assemblyconfig* will revert any previously configured resources back to their unconfigured state.

Access Rights

- *admin* – admin users have full access rights to all assemblyconfig instances in their namespace.
- *user* – users have read-only access rights to all assemblyconfig instances in their namespace.

Audits

Audits are objects created automatically by the system to track the modification of other resources in the system. A type can be defined with the "audited" flag to denote that all inserts, updates and deletes on that type should be audited. By default, operations on the following system resources will be audited:

- namespaces
- users
- profiles
- nodes
- sources
- tokens

The additional security related events will also be audited:

- Authentication using username/ passwords
- Authorization failures

Audits are required to have the following properties:

- **entity** – A resource reference used to identify the object that was manipulated to produce the audit record
- **message** – A human readable message that will be displayed when displaying audit records

Audits can optionally have the following properties:

- **op** – The name of the operation that produced this audit record

When an audit record is created the system will assign the following additional properties to the audit record:

- **node** and **nodeUUID** – Strings used to identify the node on which the audited event occurred
- **timestamp** – The time when the audit record was created (should be within milliseconds of when the audited event occurred).
- **auditId** – A unique identifier generated to identify a specific audit record

Access Rights

- *admin* – admin users can create and delete audits..
- *user* – users can create audit records (operations they perform will create audits), but they cannot delete audits.

Audits cannot be updated.

Events

Events are generated for the following operations:

- **audit** – occurs when an audit record for a non-type resource (typically `rules` or `sources`) is generated. The event path is `/audits/<resourceName>/publish`. The data provided on the event is an object with the properties:
  - entity – a ResourceReference referring to the resource instance being audited.
  - op – the operation being audited.
  - message – the audit message.

User Defined Audits

Operations on types defined by users will be audited if the “audited” property on the type definition is set to true.

In addition, if more fine-grained auditing is required users can define their own rules which create audits. For instance, in order to audit incoming messages from a source, you could do something like:

```
RULESET AuditPublishesToMySource
// Triggering condition is on any message arriving from a source named MySource
WHEN EVENT OCCURS ON "/sources/MySource" AS myVariable
// Construct an audit object with entity, op, ars_properties and message defined
var audit = {"entity": {resource: "sources", resourceId: "MySource"},
             "op": "publish",
             "ars_properties": { "importantProp": myVariable.importantProp },
             "message": "Message received from source MySource with value: " + myVariable.toString()}
// Manually insert the audit record
INSERT audits(audit)
```

This rule would create an audit record every time a message was received from the source MySource. Audits can be created from any rule or procedure, the only requirement is that the entity and message fields must be set when the audit is created.

Captures

Captures are described in detail in the Capture Reference Guide.

Access Rights

- *admin* – admin users have full access rights to all capture instances in their namespace.
- *user* – users have read-only access rights to all capture instances in their namespace.

Catalogs

Catalogs are the resources used to create and communicate with the server’s catalog functionality. They handle creating and deleting catalogs, connecting to and disconnecting from catalogs, creating and deleting entries in a catalog, and registering and unregistering for entries in a catalog.

Catalogs have the following properties:

- **name** – The name of the catalog, as generated by the procedure `Broker.makeNodeName()` in the host namespace.
- **host** – A boolean indicating whether or not a catalog is being hosted. Setting this to true will make the current namespace a catalog host.
- **allowEdge** – A boolean indicating whether or not a hosted catalog will allow connections from the edge. Setting this to true will let edge connections use the catalog as a bridge when necessary. Setting this to false will disconnect any members that connected from the edge.
- **requestedEntries** – A list of all entries that the current namespace should be registered for in this catalog. This is an array of objects with the properties below. Note that this value cannot be edited directly, and should instead be changed using the register and unregister operations.
  - **type** – The type of the entry. Valid options are “service” and “event”. More information on these types can be found in the Service Catalog and Event Catalog
  - **operation** – The operation for this entry. The values for this can depend on *type*. Valid operations for services and events are “publish” and “subscribe”.
  - **<other values>** – Every *type-operation* combination has certain values needed to identify it. Services just require the name for both publish and subscribe. Events require the name and another value that depends on the operation, as seen here.
- **catalogNode** – The name of the node used by members to communicate with the catalog. This is automatically set when hosting or connecting, and cannot be edited by users.
- **localNode** – The name of the node used by the catalog to communicate with the catalog. This is automatically set when hosting or connecting, and cannot be edited by users.
- **namespace** – The catalog’s namespace. This is automatically set when hosting or connecting, and cannot be edited by users.

Synthetic Properties

Catalogs also have several synthetic properties, which are dynamically generated on selects and cannot be directly edited.

- **resolved** – This is a list of all the entries that have successfully resolved. It is in the same format as *requestedEntries*.
- **unresolved** – This is a list of all the entries in *requestedEntries* that have not successfully resolved. It is in the same format as *requestedEntries*.

Hosting and Connecting

Hosting, connecting, and disconnecting are all done through upserts and deletes.

Upsert

Upsert has a few special cases for catalogs. It can be used without a resource id to connect to a catalog, and is used to host a catalog or append to requestedEntries.

- Hosting a catalog – To create a catalog, you need to upsert a catalog instance with *host* set to true. Note that the name of this instance must match the name generated by `Broker.makeNodeName()`.

- Connecting to a catalog – To connect to a catalog, upsert an instance with the *uri* and *accessToken* fields set to the uri of the target catalog and an access token with permissions in that namespace, and optionally *useVQS* set to `true` if connecting from an edge namespaces. When doing this all other fields (including *name*) will be ignored, and the name of the created catalog will be given in the response.
- Adding multiple *requestedEntries* – Users cannot directly change the *requestedEntries* field. Setting the field in an update or upsert will result in any new entries being merged with the current list, and it will attempt to resolve any unresolved entries. This can be useful when moving projects from one namespace to another, or when moving entries from one catalog to another. If you wish to remove entries, you'll need to use the unregister operation.

Delete

Deleting a catalog instance will cause the namespace to disconnect from the catalog or, if it is a host, disconnect all members and delete all entries.

Status

The response for the status operation on catalogs is a JSON object with the following fields:

- `name` – The name of the catalog object.
- `connectionState` – A string describing the connection state with the host namespace. The possible values are "connected", "unconnected", and "broken" (a node exists but cannot connect or the target namespace is not hosting a catalog)

Custom Operations

Most of the interactions with a catalog occur through special operations defined for the catalog resource. All of these operations act on the "system.catalogs" resource and require the resourceId (*name*) to be specified. When using the REST API, the HTTP method is POST and the body must be formatted as `{"op": <operation>, "data": <data object>}`. Any calls using remote processing must have the object formatted in the same way. For all other cases and bindings, treat the op and object as normal for that binding.

Create Entry

Creates an entry in the catalog, that can be published or subscribed to. Note that services cannot be created through this operation, and instead must be created by registering as a publisher. The body's format depends on the type of the entry, and can be found here.

Remove Entry

Removes an entry from the catalog. Only the host has permissions to remove event entries. The body must be a JSON object formatted as below

- `type` – The type of the entry.
- `name` – The name of the entry.

Register

Register for an entry. The body is a JSON object in the same format as *requestedEntries*. The response is the given entry with an additional *error* field if an error occurred while registering, or an error if the provided entry has an invalid format. Unless the entry had an invalid format, the entry is added to *requestedEntries* regardless of success.

Unregister

Unregister from an entry. The body is a JSON object in the same format as *requestedEntries*. The response is the given entry with an additional *error* field if an error occurred while unregistering, or an error if the provided entry has an invalid format. If the entry was in *requestedEntries* it is removed regardless of success.

Resolve

Attempt to complete registration for any unresolved entries in *requestedEntries*. The response is an array of all entries that failed to resolve with an additional *error* field in each.

Relationships

The Catalog instance in a namespace may have either a *publishes* or *subscribes_to* relationship to each Catalog entry in the `requestedEntries` property.

Access Rights

- *admin* – admin users have full access rights to all Catalog instances in their namespace.
- *user* – users have no access rights to Catalog instances.

Catalog Members

Catalog member objects are used by catalog hosts to represent the member namespaces. This resource is created automatically when a namespace connects to a catalog host.

Catalog members have the following properties:

- **name** – The name of the member.
- **edgeConnection** – A Boolean indicating whether the member uses a VQS connection to communicate with the catalog. This is automatically set when the member connects, and cannot be edited by users.

- **memberNode** – The name of the node used by the catalog to connect to the member. This is automatically set when the member connects, and cannot be edited by users.
- **namespace** – The member's namespace. This is automatically set when the member connects, and cannot be edited by users.

Delete

Deleting a catalog member resource forcibly disconnects the member namespace from the catalog.

Status

The response for the status operation on catalog members is a JSON object with the following fields:

- `name` – The name of the catalog member.
- `connectionState` – A string describing the connection state with the member namespace. The possible values are "connected", "unconnected", and "broken" (a node exists but cannot connect or the member namespace is not connected to the catalog)

Access Rights

- *admin* – admin users have full access rights to all Catalog Member instances in their namespace.
- *user* – users have no access rights to Catalog Member instances.

Configurations

Configuration resources are described in detail in the Configuration and Deployment Guide.

Debug Configs

Debug Configurations are used to configure logging levels and turn tracing on for rules and procedures. When a procedure or rule is executed all of the debug configurations which refer to the rule or procedure are merged to determine the minimum logging level to allow and whether or not to enable tracing. If competing debug configurations specify different log levels for the same resource, the most verbose logging level is used. Debug configurations will be automatically generated if a logging level is set or tracing is enabled from the rule or procedure editing pane in the IDE.

Debug Configurations can be created using the standard resource API patterns:

```
Method: POST
URL: /api/v1/resources/debugconfigs
Body: { "name": "MyDebugConfig",
        "type": "log",
        "value": "DEBUG",
        "resources": [
            "/rules/MyRule",
            "/procedures/MyProc",
            "/procedures/MyProc2",
            ...
        ],
        "isGlobal": false,
        "expiresAt": "2019-10-25T06:04:48.112Z"
    }
```

Debug Configuration properties include:

- **name** – A unique, human-readable name, which identifies the debug configuration
- **type** – Either "log", "trace", or "profile", refers to whether the debug configuration species a log level, enables tracing, or enables profiling data (for Grafana dashboards).
- **value** – For "log" configurations value must be one of "TRACE", "DEBUG", "INFO", "WARN", or "ERROR". For "trace" and "profile" configurations value should be "enabled", anything else will be interpreted as "disabled"
- **resources** – an array of resource references to each of the rules and procedures which the debug configuration applies to.
- **isGlobal** – A Boolean that, when true, indicates the debug configuration applies to all rules and procedures. This property cannot be set to true if resources are specified in the resources array.
- **expiresAt** – A DateTime property that specifies when the debug configuration should be automatically deleted. If not specified on insert, the value is set to 2 hours after the creation time.

Access Rights

- *admin* – admin users have full access rights to all Debug Configuration instances in their namespace.
- *user* – users have read-only access rights to all Debug Configuration instances in their namespace.

Delegated Requests

A delegated request represents a resource operation whose execution is deferred until the submission of an associated, one time use, code value. The user who creates a delegated request is known as the "creator" and the user who triggers its execution is known as the "processor" (these may or may not be the same user). Delegated requests can also specify one or more VAIL "scripts" (which consist of a list of VAIL statements) which will be run after completion of the initial operation. Depending on the definition of the request this execution can occur using the security context of the creator or the processor.

Currently the use of delegated requests is restricted to the Vantiq platform and they cannot be created by users (regardless of their authorizations). They are primarily used to perform on-boarding of new users and to grant new authorizations to existing users via the “invite” system (the links contained in an invite email trigger execution of a system created delegated request).

Custom Operations

Publish

Sends an existing delegated request to an email address–when executed, will grant the executing user access to the namespace. For REST this is a POST directly to the `api/v<version>/resources/delegatedrequests` URI. The argument to this operation is a JSON document with the following properties:

- `op` – “publish”
- `data` – a JSON document with the following properties:
- `to` – the address to which the message will be sent. Required.
- `source` – the name of the source that will send the request. Optional – if not specified the installation’s default email source will be used.
- `callbackUri` – the url to which the recipient will be redirected to once the request is accepted. Optional – if not specified the recipient will not be redirected.
- `from` – the sender address of the message. Optional – if not specified then a default value will be used.
- `parameters` – an object whose key-value pairs will be used to fill out the email template. Optional – the default templates do not require parameters, and any not specified here will be pulled from the delegated request instance. More information on parameters can be found here.
- `htmlTemplate` – a string that is the name of the Document to use as a template. Can only find Documents in the current namespace. Optional – if not set then it will use the template from the delegated request or the default template if the delegated request does not specify. More information on template formatting can be found here.
- `subject` – a string that will be the subject of emails sent by the request. If not specified and not part of a namespace invite, it will default to “Notification from Vantiq”.

Documents

A document represents a file stored in the Vantiq Database. Document instances are created by uploading the contents of the document using a multi-part MIME message of the following form:

```
POST https://dev.vantiq.com/api/v1/resources/documents
<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: <contentType>
<file content>
<your boundary marker>
```

where:

- `<assignedName>` is the name of the document as stored in **documents** instance.
- `<fileName>` is the name of the file from which the content was obtained.

In addition, the appropriate HTTP headers must be set:

```
ContentLength: <length of http content in above message>
ContentType: multipart/form-data; boundary=<your boundary marker>
```

Once the file has been uploaded its meta-data will be available via the resource URI: `api/v1/resources/document/<assignedName>`.
This will return the following JSON representation:

```
{
    "name": "assets/img/autopsy/statement1.png",
    "fileType": "image/png",
    "content": "/docs/assets/img/autopsy/statement1.png"
}
```

The `content` property is the absolute URI that can be used to retrieve the file’s content from the Vantiq server. The `fileType` property is set by Vantiq based on the uploaded file’s extension, but it can be changed once the file has been uploaded (this will influence whether the file is treated as text or binary data, so care must be taken when altering this value).

Vantiq uses documents to store the system documentation available on the Vantiq IDE. Vantiq reserves the `system/` document name prefix for this purpose which means that no user created documents may begin with that prefix. Within the user’s namespace the `public/` prefix is used to indicate that the document contents should be available without requiring an authenticated user via the URL `/ui/docs/NS/<namespace>/<relativePath>` where:

- `<namespace>` is the namespace from which to obtain the document.
- `<relativePath>` is the document name without the `public/` prefix.

For example, if you upload a document under the path `public/images/myImage.png` in the namespace “example”, then it will be accessible without authentication via the URI `https://dev.vantiq.com/ui/docs/NS/example/images/myImage.png`.

The document name can be any string, so long as the first character is not a `/`.

Note that the resource instance URI and the `/docs` URI always require authentication, even for public documents.

Copy

To copy a Document use either the upsert or duplicate operation. The upsert operation copies a document from a target Namespace (pull) while the duplicate operation copies a Document to one or more target Namespaces (push). Both operations can be used within the same namespace and are in that case semantically equivalent.

Upsert

A Document can be copied from a target Namespace by issuing a POST of the following form:

```
Method: POST
URL: /api/v1/resources/documents
Body: { "name": "<copyName>",
        "fromName": "<fromName>",
        "fromNode": {
            "<nodeName>"
        }
    }
```

where:

- `<copyName>` is the name of the new Document.
- `<fromName>` is the name of the existing Document to copy.
- `<nodeName>` is the Node name specifying where the Document to copy is located.

Because a Document can be copied only from a single location, `<nodeName>` is a Node name, therefore resolving to a single Node. If `<nodeName>` is not specified, the Document is copied within the same Namespace.

Create Document from Temp Blob

If you have obtained a `TempBlob` (*e.g.*, from a VIDEO or REMOTE source), you can create a document from that information. To do this, use the `fromTemp` property, specifying the Resource Reference of the Temp Blob. Due to the nature of Temp Blobs (resident in memory), the `fromNode` parameter is not supported.

```
Method: POST
URL: /api/v1/resources/documents
Body: { "name": "<copyName>",
        "fromTemp": "<Temp Blob reference>"
      }
```

Duplicate

A Document can be copied to one or more target Namespaces by issuing a POST of the following form:

```
Method: POST
URL: /api/v1/resources/documents/<resourceId>
Body: { "op": "duplicate",
        "data": {
            "duplicateName": "<copyName>",
            "remoteNodes": {
                "<nodeQueryConstraint>"
            }
        }
    }
```

where:

- `<copyName>` is the name of the new Document.
- `<resourceId>` is the name of the existing Document to copy.
- `<nodeQueryConstraint>` is a query constraint defining the set of Nodes where the Document must be copied to.

The Document is copied to all Nodes that `<nodeQueryConstraint>` resolves to. If `<nodeQueryConstraint>` is not specified, the Document is copied within the same Namespace.

To duplicate a document using a WebSocket or VAIL binding, issue a request of the following form:

```
{
  "op": "duplicate",
  "resourceId": "<resourceId>",
  "resourceName": "documents",
  "object": {
    "duplicateName": "<copyName>",
    "remoteNodes": {
      "<nodeQueryConstraint>"
    }
  }
}
```

For an upsert operation, any Node referenced in `<nodeName>` must be defined with an HTTP URI scheme. For a duplicate operation, any Node referenced in `<nodeQueryConstraint>` can be defined with an HTTP or WebSocket URI scheme.

Access Rights

- *admin* – admin users have full access rights to all Document instances in their namespace.
- *user* – users have full access rights to any Document instance which they have created.

Examples

Create or Update Document

```
POST https://dev.vantiq.com/api/v1/resources/documents
<your boundary marker>
Content-Disposition: form-data; name="sampleDocument"; filename="sourceFile.html"
Content-Type: text/plain
This is the actual two lines of text
that is being upload as the content of the document.
<your boundary marker>
```

The headers:

```
ContentLength: 209
ContentType: multipart/form-data; boundary=dLV9Wyq26L_-JQxk6ferf-RT153LhOO
```

Delete Document

```
DELETE https://dev.vantiq.com/api/v1/resources/documents/sampleDocument
```

Event Generators

Event Generators allow Vantiq developers to simulate data from any Type, Topic, or Source.

An Event Generator is defined by:

- **name**: A unique name for the generator
- **events**: a list of event descriptors defining the events to be generated

An Event Descriptor defines one or more events that will occur on a specified resource, and how each event message will be generated. Each Event Descriptor defines:

- **resource** (ResourceReference) *required*: Reference to the resource that will generate the event
- **op** (String) *required for type resources*: Type of operation (INSERT, UPDATE, DELETE) of the event if applicable
- **ruleList** (List of Objects) *required*: List of Generator Rules that will determine how each event message is constructed
- **qual** (Object) *required for update/delete events*: Qualifier object describing which instances should be updated or deleted
  For Example: *{name: "abc"}* would apply to all instances where name == "abc"
- **startAfter** (Integer): Duration (milliseconds) before this descriptor begins producing events (after Event Generator begins running).
- **periodic** (Boolean): true if this is a repeated event. Periodic events must specify either a duration or number or iterations as well as an interval
- **duration** (Integer): Duration (milliseconds) that this event descriptor should actively produce events. The total number of produced events will be equal to *duration/interval*.
- **iterations** (Integer): Number of events that this Event Descriptor will produce
- **interval** (Integer) *required if periodic*: Duration (milliseconds) between periodic events produced by the Event Generator.
  Note: Either the Duration or Iterations property may be set but not both. If either is set, the interval property is required.

See the Event Generator Documentation for running a generator and stopping a generator.

Standard Operations

Status

The response for the status operation for an event generator will reveal whether the generator is currently running. The response will be an object with the following properties:

```
{
    <GeneratorName>: [
        {
            _id: <runningGeneratorId if currently running>,
            startTime: <runningGeneratorStartTime if currently running>
        }
    ]
}
```

Relationships

The event generator instance has a *dependency* relationship for every Type, Topic, and Source on which the generator produces events.

Access Rights

- *admin* – admin users have full access rights to all eventgenerator instances in their namespace.
- *user* – users have no access rights to any eventgenerator instances in their namespace.

Groups

A group is a set of users and access tokens with shared access to a collection of resources. Instances of types can be defined with an ars_group field which restricts access to the record to only members of the group. Members must be defined as resource references defining the resource (either `tokens` or `users`) and the resourceId specifying the username or token name.

Types that are going to have group-restricted access need to be defined with the groupEnabled flag set to true on the type definition. This flag will add the extra qualifications to any query performed on the “grouped” type. The groupRequired flag can be set on the type definition as well to indicate that all inserts on the type must specify a non-null value for ars_group. Groups can be enabled without being required, in which case records with a null ars_group value will be visible to all users with access to the type.

Access Rights

- *owner* – The owner of a group is specified when the group is created and is the only user who can add members to the group or delete the group
- *member* – Members of the group have access to records tagged with an ars_group value of the group name. Members can remove themselves from a group, but not add other users to the group.
- *admin* – The namespace admin can see all records in the namespace, regardless of groups.
- *users* – General users can see which groups exist, and see who owns the group, but can not access any of the elements tagged with an ars_group value of the group name.

Examples

Create a group

```
POST https://dev.vantiq.com/api/v1/resources/groups
{ "name": "myGroup",
  "owner": "myUsername",
  "members": [
    {
      "resource: "users",
      "resourceId": "myUsername"
    },
    {
      "resource": "users",
      "resourceId": "anotherUser"
    },
    {
      "resource": "users",
      "resourceId": "aThirdUser"
    }
  ]
}
```

Add a member to the group

```
PATCH https://dev.vantiq.com/api/v1/resources/groups/myGroup
{ "op": "add",
  "path": "/members/-",
  "value": {
    "resource": "users",
    "resourceId": "newUser"
  }
}
```

Adding members can also be done via POST

```
POST https://dev.vantiq.com/api/v1/resources/groups/myGroup
{
  "members": [
    {
      "resource: "users",
      "resourceId": "myUsername"
    },
    {
      "resource": "users",
      "resourceId": "anotherUser"
    },
    {
      "resource": "users",
      "resourceId": "aThirdUser"
    },
    {
      "resource": "users",
      "resourceId": "aFourthUser"
    }
  ]
}
```

The list provided via a POST will overwrite the existing list of members, so be sure to include any existing users in the members list when adding more users.

Remove a member from the group

```
PATCH https://dev.vantiq.com/api/v1/resources/groups/myGroup
{ "op": "remove",
  "path": "/members/-",
  "value": {
    "resource": "users",
    "resourceId": "newUser"
  }
}
```

Remove members with a POST

```
POST https://dev.vantiq.com/api/v1/resources/groups/myGroup
{
  "members": [
    {
       "resource: "users",
       "resourceId": "myUsername"
     },
     {
       "resource": "users",
       "resourceId": "aFourthUser"
     }
   ]
 }
```

This would remove “anotherUser” and “aThirdUser”.

It is not possible to remove the owner from a group, so the owner can safely be omitted from the members list without risk of dropping the owner from the group.

Images

An image represents image content stored in the Vantiq Database. Image instances are created by uploading the contents of the image using a multi-part MIME message of the following form:

```
POST https://dev.vantiq.com/api/v1/resources/images
<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: <contentType>
<file content>
<your boundary marker>
```

where:

- `<assignedName>` is the name of the image as stored in **images** instance.
- `<fileName>` is the name of the file from which the content was obtained.
- `<contentType>` is the mime type of the image (*e.g.* `image/jpeg` , `image/png`)

In addition, the appropriate HTTP headers must be set:

```
ContentLength: <length of http content in above message>
ContentType: multipart/form-data; boundary=<your boundary marker>
```

Once the file has been uploaded its meta-data will be available via the resource URI: `api/v1/resources/images/<assignedName>`.
This will return the following JSON representation:

```
{
    "name": "assets/img/cars/mycar.png",
    "fileType": "image/png",
    "content": "/pics/assets/img/cars/mycar.png"
}
```

The `content` property is the absolute URI that can be used to retrieve the file's content from the Vantiq server. The `fileType` property is set by Vantiq based on the uploaded file's extension, but it can be changed once the file has been uploaded.

The image name can be any string, so long as the first character is not a `/`.

Note that the resource instance URI and the `/pics` URI always require authentication, even for public images.

Create Image from Temp Blob

If you have obtained a `TempBlob` (*e.g.*, from a VIDEO or REMOTE source), you can create an image from that information. To do this, use the `fromTemp` property, specifying the Resource Reference of the Temp Blob.

```
Method: POST
URL: /api/v1/resources/images
Body: { "name": "<copyName>",
        "fromTemp": "<Temp Blob reference>"
      }
```

Access Rights

- *admin* – admin users have full access rights to all Image instances in their namespace.
- *user* – users have full access rights to any Image instance which they have created.

Examples

Create or Update Image

```
POST https://dev.vantiq.com/api/v1/resources/images
<your boundary marker>
Content-Disposition: form-data; name="sampleImage"; filename="mycar.png"
Content-Type: image/png
<bytes for the image>
<your boundary marker>
```

The headers:

```
ContentLength: Varies, based on boundary & image bytes length
ContentType: multipart/form-data; boundary=dLV9Wyq26L_-JQxk6ferf-RT153LhOO
```

Delete Image

```
DELETE https://dev.vantiq.com/api/v1/resources/images/sampleImage
```

K8s Clusters

A K8s Cluster defines a Kubernetes cluster that is known to Vantiq. This is a Kubernetes cluster to which Vantiq is given access, and it may have K8s Installations deployed. It is not necessarily a Kubernetes cluster in which the Vantiq system is running.

K8s Clusters are created using the usual DDL operations. To create/update a K8s Cluster, POST the following JSON document to the resource URI:

```
{
  "name": "<name by which Vantiq will know the cluster>",
  "ingressDefaultNode": "<node name for access to installations with inbound ports>"
}
```

where

- The **name** property is the name by which Vantiq users will refer to this cluster
- The **ingressDefaultNode** is the default name of the DNS node used to route to any K8s Installation with an `inboundPort`, when that K8s Installation's `inboundPort` does not include a `host` specification.
    - If the value is not provided, Vantiq will generate a name based on the cluster name itself. This value is a default value only.
    - It is up to the system administrators to define DNS entries (in `/etc/hosts` or local DNS servers as appropriate) for any hosts used in `inboundPort` definitions (either explicitly or the `ingressDefaultNode` used by default).
    - This value is used for routing to a K8s Installation with an `inboundPort`. If the default value is not used, no DNS routing is necessary.
    - Each K8s Installation with an `inboundPort` should use its own host value. Please see Regarding Host Names for Inbound Port Specifications for more information.
- The **lifecycleOperationCount** is an approximate count of the number of operations performed on this K8sCluster.

Once created the K8sCluster resource has the following JSON representation:

```
{
  "name": "<name by which Vantiq will know the cluster>",
  "ingressDefaultNode": "<Default DNS name on which Ingress operations are defined>"
  "lifecycleOperationCount": <number of operations performed on this cluster>
}
```

If you've created a cluster named *aCluster* with an **ingressDefaultNodeName** of *aClusterNode*, the resulting K8sCluster would look like this:

```
{
  "name": "aCluster",
  "ingressDefaultNode": "aClusterNode"
  "lifecycleOperationCount": 14
}
```

(assuming that about 14 operations have been completed).

To delete a K8sCluster, use the DELETE method on the resource URI. Note that it is an error to DELETE a K8sCluster that has K8s Installations or K8s Workitems outstanding. To delete a K8sCluster, please delete the K8s Installations first.

Custom Operations

K8s Clusters support the following custom resource operations. Each of these operations returns an K8s Workitem. These operations are asynchronous – the call creates and enqueues the request to perform some action on the associated K8sCluster. A worker service will pick up the request and process it, and the completion of that request will update the data model (*e.g.* cause the installation to be created). There may be some delay before this happens.

Deploy

Causes the creation of a K8s Installation running an image in this cluster. It is assumed that appropriate resources exist in the cluster – Vantiq is not involved in that work. Note also that performing a `deploy` operation on an existing K8s Installation will update that installation with the new configuration.

To perform the deployment, POST a request to the resource URI with the suffix "/command". (Note that the following presumes some familiarity with Kubernetes.) The argument to this operation is a JSON document with the following properties:

- `op` – "deploy"
- `data` – a JSON document with the following properties:
    - `name` – The name to be assigned to the running instance of this deployment. This will also be the name of the K8s Installation in Vantiq.
    - `k8sNamespace` – the Kubernetes namespace in which the set is to be created.
    - `config` – Any configuration information required by the image. The `config` property is a List of JSON objects with the following properties, where each object has a `type` property, and other properties depending upon their type. One instance with type `image` is required. All other types are optional.
        - `type`: `image` – (exactly one instance of this is required.)
            - `name`: name of the image for the Kubernetes system to fetch. May include a repo name.
            - `imagePullSecret`: name of the Kubernetes secret necessary to pull the image from its repo, if any. This may be required if the repository from which the images is to be pulled is not public.
            - `serviceAccount`: name of the Kubernetes service account under which the installation will run. Note that if the service account has the appropriate image pull secret associated with it, the `imagePullSecret` entry here is not necessary.
            - `replicaCount`: number of replicas the installation will contain. The default value is 1, and that number should not be changed unless the image being deployed is specifically designed to support multiple replicas.
        - `type`: `file` – defines a file and content to be added to the installation
            - `path` – where in the instance the file is to be mounted
            - `filename` – the name of the file to place at the path
            - `dataSource` – the Kubernetes resource type that will be created or referenced, must be one of the following: `configMap` or `secret` (if not specified, will default to `configMap`)
            - File Content – file content can be specified a number of ways using different sets of properties. These sets are mutually exclusive.
                - Content provided directly
                    - `content` – the contents of the file. We support only text files.
                - Content obtained from an existing Kubernetes ConfigMap or Secret that is already present in the Kubernetes namespace to which you are deploying

        - `name` – name of an existing ConfigMap or Secret (in the same Kubernetes namespace) from which to obtain the content
        - `key` – the ConfigMap or Secret key referencing the desired content.
      - Content from a Vantiq document, image, or video
        - `resourceType` – the Vantiq resource type – must be one of
          - `documents`
          - `images`
          - `videos`
        - `resourceName` – name of an existing Vantiq resource. Note that in this case, the `filename` can be omitted. If no `filename` is provided, we will use the last part of the `resourceName` as the name of the file. That is, if the `resourceName` is `document/name.txt`, `name.txt` will be used as the `fileName`.
        - `treatAsText` – (optional) treat the resource as a text resource when providing the data to Kubernetes. Kubernetes mounts data differently for text vs. binary data. This is necessary when the Vantiq document does not have a `fileType` that starts with `text`.
- `type`: `inboundPort` – defines an inbound port to be opened on behalf of this installation. This would be the case when a connector acts as a server (for example, the UDP Source). Properties as follows (required unless otherwise specified).
  - `host` – name of host for this port. This specifies that anything directed to this host name will be directed to this inbound port. This host must resolve to the cluster address for the K8s cluster in question. If this is missing, the K8sCluster's `ingressDefaultNode` is used.
  - `port` – (required) integer specifying the port number expected to be opened This is the port the image is expecting to use.
  - `protocol` – (required) Kubernetes supported protocol (TCP (includes HTTP/HTTPS), UDP)
  - `portName` – (optional) name to be assigned to the port
  - `serviceType` – (optional) the K8s service type. Defaults to "NodePort"
  - `serviceName` – (optional) a name to give the service. We generate a reasonable one, but you can override it.
  - `clusterIP` – (optional) IP address to be assigned to the created pod. This should be very rarely used.
- `type`: `environmentVariable` (optional) – this indicates that an environment variable with the indicated name and value should be in place when the image runs.
  - `name` – the name of environment variable
  - `value` – the value to be assigned to the environment variable. To provide the value from a Vantiq Secret, use the construct `@secrets(VantiqSecretName)`, where `VantiqSecretName` is the name of the Vantiq Secret from which to obtain the value. If this construct is used, the value will be stored as a Kubernetes secret (that is, the `asSecret` value will be set to `true`). If you wish to set the environment variable's value to "@secrets(someSecretName)", place a backslash (`\`) before the `@secrets` string.
  - `asSecret` – boolean indicating whether this value should be stored as a Kubernetes secret
  - `secretName` – name of the Kubernetes secret from which the value is to be taken. If the `secretName` value is present, the value of that Kubernetes secret will be used (with the key named by the `key`). If the `secretName` is not present, a new Kubernetes secret will be created and the value stored there. The properties `value` and `secretName` are mutually exclusive. That is, if you are using an existing Kubernetes secret, the setting of the value is not permitted.
  - `key` – name of the key within the Kubernetes secret named by `secretName` from which to extract the value.
- `type`: `persistentVolumeRef` (optional) – this indicates that the installation will refer to an existing volume in the cluster to which it is being deployed
  - `name` – name of the volume to which to refer.
  - `path` – path in the installation at which to mount the named volume
- `type`: `annotation` (optional) – this indicates that the deployed installation should have this K8s annotation
  - `name` –- name of the annotation
  - `value` –- Value to give the annotation
  - `targetResource` (optional) – the Kubernetes resource type to which this annotation should apply. This will add the annotation to the Kubernetes resources created as part of this K8s Installation. Valid values are `configmap`, `ingress`, `pod`, `secret`, `service`, and `statefulset`. If this property is absent, the annotation will apply to all Kubernetes resources created as part of this K8s Installation.
- `type`: `label` (optional) – this indicates that the deployed installation should have this Kubernetes label applied
  - `name` –- name of the label
  - `value` –- value to give the label
  - `controlsRouting` (optional) – -a boolean value indicating whether this label should be used to link the service and underlying stateful set. This means that this label is used to *route* messages to the deployed entity. (Note: this behavior is primarily of interest to `NodePort` services.) If no labels are specified as routing controllers, the system will generate something appropriate. The value here can be a boolean or string value. If supplied as a string, any value not interpreted as true will be considered false.
  - `targetResource` (optional) – the Kubernetes resource type to which this label should apply. This will add the label to the Kubernetes resources created as part of this K8s Installation. Valid values are `configmap`, `ingress`, `pod`, `secret`, `service`, and `statefulset`. If this property is absent, the label will apply to all Kubernetes resources created as part of this K8s Installation.
- `type`: `hostAlias` (optional) – a name to IP address mapping to be configured for this installation
  - `name` – the host name to define
  - `hostIP` – the IP address to supply
- `type`: `probe` (optional) adds a Kubernetes *probe* definition to the installation. Kubernetes *probes* are used to help Kubernetes determine when your installation has completed its startup functions (a `startup` probe), whether the installation can accept requests (a `readiness` probe), or whether the installation is still functional (a `liveness` probe).
  - `probeType` – one of `startup`, `readiness` or `liveness`
  - `initialDelaySeconds` – the number of seconds to wait before running this probe
  - `periodSeconds` – how often to run the probe
  - `timeoutSeconds` – (optional) number of seconds after which the probe times out. If not provided, Kubernetes defaults apply.

- `successThreshold` – (optional) number of successful probe attempts to consider the installation operational. If not provided, Kubernetes defaults apply.
- `failureThreshold` – (optional) number of failed probe attempts required to consider the installation failed. A failed installation will be terminated and restarted (by Kubernetes). If not provided, Kubernetes defaults apply.
- `requestType` – one of `command`, `http`, or `tcp`. Based on the request type, the following additional properties are specified
  - `requestType`: `command`
    - `commands` – a list providing the command and parameters (list of strings). For example, if an installation created a file named `/installation/up` when it was ready, a `commands` entry might be `['cat', '/installation/up']`. This would have Kubernetes execute the command `cat /installation/up` to run that probe.
  - `requestType`: `tcp`
    - `port` – the name or integer port number to which to connect. You can specify this as the numeric port number OR the `portName` defined as part of an `inboundPort` specification.
  - `requestType`: `http`
    - `path` – the path to use in constructing the URL for making this probe
    - `port` – the port to be used in constructing the URL for making this probe. As with the `tcp` probe above, you can use the `portName` associated with an `inboundPort`.
    - `headers` – a set of key/value pairs sent as part of the request. Both the key and the value must be strings.
- `type`: `resourceRequest` – specifies the minimum requirement for the following resources, and
- `type`: `resourceLimit` – specifies the absolute maximum for the following resources
  - The following resources are used to enumerate both the `resourceRequest` and `resourceLimit` areas. They are specified using Kubernetes resource terms. For example, to request one half of a CPU, use `500m` meaning *500 millicpus*. See the Kuberenetes documention for further details.
  - `cpu`: the number of CPUs for the deployed installation
  - `memory`: the amount of memory for the deployed installation
  - `nvidia.com/gpu`: the number of NVidia GPUs required
  - `amd.com/gpu`: the number of AMD GPUs required.
- `type`: `hardAffinity` – specifies a key/values pair that is set as the Kubernetes Node Selector for the deployed pod. To use this, consult your Kubernetes administrator. Note that you may see these added to installations deployed to the `self` cluster. This is part of Vantiq's management of the cluster in which it operates.
  - `key` – The *key* value to which the `values` will apply.
  - `values` – A list of values for the `key` specified that are acceptable. The node selector will be generated using the *in* operator (meaning that nodes whose `key` value is contained in this list are valid for use for this installation).
    - Note that you must take care with this specification. If the `key` or `value` is misspecified, generally no error will result. Instead, Kubernetes will wait for an appropriate node to appear.
- `configRef` – Instead of the `config` property, the `configRef` property can be provided. This will be a Resource Reference to a document whose contents contain what would be in the `config` property as described above.

This operation returns the K8s Workitem for the request. Note that this is the only way to create or update a K8sInstallation.

Regarding Host Names for Inbound Port Specifications

If an installation has an inbound port, that installation will be available at URL http://*host*:80, assuming the cluster is set up correctly. The *host* here will be either the `host` value from the `inboundPort` configuration item or, if no such value is provided, the K8s Cluster's `ingressDefaultNode` value. You should *always* provide the `host` property here; each K8s Installation that has an `inboundPort` should have its own value for `host`. That host name is used by Kubernetes to route requests to the installation.

Moreover, as noted, the host name given (or the K8s Cluster's `ingressDefaultNode` if that is used) must resolve to the cluster address for the Kuberetes cluster in question. That is, the URL `http://host:80` must be a valid, resolvable URL. Requests sent to the Kubernetes cluster using that host name will be routed to the K8s Installation specified. It is up to the system administrators to define DNS entries (in `/etc/hosts` or local DNS servers as appropriate) for any hosts used in `inboundPort` definitions (either explicitly or throught the use of the `ingressDefaultNode` used by default).

(Please see the External Lifecycle Management Guide for more information concerning defining K8sClusters).

FetchLogs

Fetches the pod logs for an installation. A *pod* is the K8s reference for the running component(s) of an installation. Thus, this operation fetches the log output of the installation, storing it as a Vantiq document (or collection thereof). Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "fetchLogs"
- `data` – a JSON document with the following properties:
  - `name` – The name of K8s Installation for which to fetch the logs.
  - `podName` – (optional) The name of the pod for which to fetch logs. If not present, this will fetch all logs for the installation. (This parameter is not often used.)
  - `documentGroup` – (optional) Where to upload the logs.

All pod logs are uploaded as Documents in the namespace in which the Vantiq K8sCluster is found, with the logs named with the document group followed by the pod name with a timestamp. The individual pod logs are named with the pod name followed by a timestamp. The default document group name will include the name of the cluster and the installation name.

For a cluster named *cluster* and an installation named *inst*, a pod log for an installation named *jdbc* will be uploaded to the document named

```
cluster/inst/jdbc-0-2021-07-14T12-13-14.000-0700.txt
```

if the *fetchLogs* request was processed at exactly 14 seconds after the 12:13 on the 14 July 2021 (Pacific Daylight Time).

Assuming the same installation and time of request execution, if the `documentGroup` is set to *my/group*, the pod log will be uploaded to

```
my/group/jdbc-0-2021-07-14T12-13-14.000-0700.txt
```

This operation returns the K8s Workitem for the request.

Restart

Restarts a k8s installation (as above). As with *deploy*, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "restart"
- `data` – a JSON document with the following properties:
  - `name` – The name of K8s Installation to restart.

This operation returns the K8s Workitem for the request.

Shutdown

Shuts an installation down. Send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "shutdown"
- `data` – a JSON document with the following properties:
  - `name` – The name of K8s Installation to restart.

This operation returns the K8s Workitem for the request.

Undeploy

Removes an installation from its K8s Cluster and from the Vantiq system. Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "undeploy"
- `data` – a JSON document with the following properties:
  - `name` – The name of K8s Installation to undeploy.

This operation returns the K8s Workitem for the request.

Access Rights

- *admin* have the rights to modify the K8s Clusters.
- *users* have no access to K8s Clusters.

Examples

Deploy

```
POST https://dev.vantiq.com/api/v1/resources/k8sclusters/aCluster/command
{
    "op": "deploy",
    "data": {
        "name": "SourceDeployment",
        "k8sNamespace": "myNamespace",
        "config": [
                    {
                      "type": "image",
                      "name": "myRepo/connectorImage"
                    },
                    {
                      "type" : "file",
                      "content" : "some data",
                      "path" : "/app/somedirectory",
                      "filename": "myfile",
                    }
                  ]
    }
}
```

Note that the `config` data will be image specific. Each image may require different data. It is up to the deployer to understand the image requirements.

This operation will return the K8s Workitem for the request created. For the operation performed above, this will look like the following:

```
{
   "requestId": "<some UUID>",
   "clusterName": "aCluster",
   "operation": "deploy",
   "timestamp": <time of request>
   "status": "REQUESTED"
   "request": <config value from request>
   "requestedBy": "<name of the Vantiq user who performed the deploy>"
}
```

Restart

```
POST https://dev.vantiq.com/api/v1/resources/k8sclusters/aCluster/command
{
    "op": "restart",
    "data": {
        "name": "SourceDeployment"
    }
}
```

K8s Installations

K8s Installations represent Kubernetes installations (Stateful Sets with services, etc.) in a Kubernetes Cluster that have been deployed by Vantiq. K8s Installations are created using the *Deploy* operation on a K8s Cluster.

The representation of a K8s Installation is as follows,

```
{
   "name": "SourceDeployment",
   "clusterName": "aCluster",
   "installationDate": "Date & Time of deployment"
   "installationStatus": statusValue,
   "installedBy" : <name of user who performed the deploy>
   "replicaCount": <number of instances>
   "configSpec": <specification used to create the installation>
   "k8sState": <information about the installation as reported by Kubernetes>
   "k8sNamespace": "myNamespace",
   "k8sInstallationInfo": Object containing information about the Kubernetes deployment
   "k8sInstallationType": generally STATEFUL_SET
   "k8sInstallationIdentifier": <Kubernetes identifier>
   "k8sAssociatedServices": <List of services created to support the installation>
   "k8sAssociatedIngresses": <List of ingresses created to support the installation>
   "k8sAssociatedConfigMaps":< List of config maps created to support the installation>
   "k8sAssociatedSecrets": <List of secrets created to support the installation>
   "quotaConsumed": <Object containing the quota consumed by this installation>
}
```

where

- **name** is the name of the installation
- **clusterName** is the name of the cluster within which this installation is running
- **installationDate** is the date/time of the most recent deployment attempt
- **intallationStatus** is Vantiq's view of the it's interactions concerning this installation
- **installedBy** is the name of the Vantiq user who performed the deploy operation
- **replicaCount** is the number of replicas
- **configSpec** is the configuration used to create the installation
- **k8sState** is the current state of the installation as reported by Kubernetes
- **k8sNamespace** is the name of the Kubernetes namespace within the cluster
- **k8sInstallationInfo** general Kubernetes information about the installation
- **k8sInstallationType** Kubernetes component type for installation. Currently STATEFUL_SET.
- **k8sInstallationIdentifier** Identifier proved by Kubernetes
- **k8sAssociatedServices** List of Kubernetes services created to support this installation
- **k8sAssociatedSecrets** List of Kubernetes secrets created to support this installation
- **k8sAssociatedConfigMaps** List of Kubernetes config maps created to support this installation
- **k8sAssociatedIngresses** List of Kubernetes ingresses created to support this installation
- **quotaConsumed** For installations in the Vantiq system `self` cluster, the quota consumed by this installation. This may have values for **vCPU**, **memory**, **gpuAmd**, and/or **gpuNvidia**. This will be present only for installations deployed to the Vantiq system's *self* cluster

The **installationStatus** is a string containing one of the following values:

- STARTING – Vantiq has asked K8s to restart the deployment
- STOPPING – Vantiq has asked K8s to shutdown the deployment
- RUNNING – The deployment has succeeded and the deployment is operating
- FAILED – The last operation (deploy, restart, shutdown) failed

- STOPPED – The shutdown operation succeeded
- DAMAGED – The installation has been altered in an unexpected way. This might indicate that some component of the installation has been removed.

Custom Operations

K8s Installations support the following custom resource operations.

FetchLogs

Fetches pod logs from the installation, storing them in documents. Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "fetchLogs"
- `data` – an JSON document with the following optional properties
  - `podName` – (optional) The name of the pod for which to fetch logs. If not present, this will fetch all logs for the installation. (This parameter is not often used.)
  - `documentGroup` – (optional) Where to upload the logs.

If neither property is provided, the `data` property will be an empty JSON document.

This operation returns the K8s Workitem for the request.

Restart

Restarts an installation (as above). Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "restart"
- `data` – an empty JSON document

This operation returns the K8s Workitem for the request.

Shutdown

Shuts down an installation. Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "shutdown"
- `data` – an empty JSON document

This operation returns the K8s Workitem for the request.

Undeploy

Removes an installation from its k8s cluster & from Vantiq. Again, send the following to the resource URI with "/command" appended to the end. The argument to this command is a JSON document with the following properties:

- `op` – "undeploy"
- `data` – an empty JSON document

This operation returns the K8s Workitem for the request.

Access Rights

- *admin* have the rights to view the K8s Installations.
- *users* have no access to K8s Installations.
- No direct update operates are allowed.
  - Updates are performed only using the custom operations described above.

Examples

Shutdown

```
POST https://dev.vantiq.com/api/v1/resources/k8sinstallations/someDeployment/command
{
    "op": "shutdown",
    "data": {}
}
```

K8s Workitems

K8s Workitems represent Kubernetes work to be performed on a K8s Cluster. K8s Workitems result from the various command operations performed on a K8s Cluster or K8s Installation.

The representation of a K8s WorkItem is as follows,

```
{
   "requestId": "<some UUID>",
   "clusterName": "<name of the K8sCluster to which the work item is directed",
   "operation": "<operation name>",
   "timestamp": <time of request>
   "status": "<operation status>"
   "request": <config value from request>
   "requestedBy": "<name of the Vantiq user who performed the deploy>"
   "ancillaries": <Object describing additional items required to perform the work>
}
```

where

- **requestId** is the unique identity of the work item
- **clusterName** is the name of the cluster within which this deployment is running
- **operation** is the operation to be performed – `deploy`, `shutdown`, `restart`, or `undeploy`
- **timestamp** is the date/time at which the work item was last altered
- **status** is Vantiq's view of status of the work item.
- **request** the `config` value from the request
- **requestedBy** is the name of the Vantiq user performing the deploy operation.
- **ancillaries** is an object containing the set(s) of Kubernetes components created to support this installation

The **status** is a string containing one of the following values:

- REQUESTED – Vantiq has asked K8s to perform the deployment
- PENDING – a worker has picked up the work
- FAILED – the operation (deploy, restart, shutdown) failed
- INVALID – a problem was found with the request

Access Rights

- *admin* have the rights to modify the K8sWorkitems.
- *users* have no access to K8sWorkitems.

LLMs

LLMs represent a Large Language Model that can either be run by the Vantiq platform or which is hosted somewhere else, but reachable via some remote protocol (most often REST over HTTP). Vantiq has preconfigured support for the following models: GPT-4, GPT-4o-mini, Open AI Embeddings and Sentence Transformers. It is also possible for customers to configure access to additional, "custom" models (contact Vantiq Support for more details).

See also built-in service io.vantiq.ai.LLM.

LLMs have the following properties:

- **name** (String) – the name of the LLM. Must be unique in a given Vantiq namespace. May include a package name.
- **type** (String) – the type of LLM. The value is immutable once set. Indicates what functions the LLM performs. Possible values are:
  - **embedding** – the LLM is used to create vector embeddings when loading a semanticindex.
  - **generative** – the LLM is used to generate responses based on user input.
- **modelName** (String) – the name of the actual model used by this LLM. The value is immutable for *embedding* models. If the model is not one that is known to the Vantiq platform, the user must supply the information necessary to configure access to the model at runtime. The recognized models are:
  - Versions 3.5 and 4 of the OpenAI models.
  - The "text-embedding-ada-002" model from OpenAI.
  - The HuggingFace Sentence Transformer models.
- **description** (String) – an optional description of the LLM.
- **config** (Object) – optional configuration used when constructing the runtime form of the LLM. Some common properties are:
  - **temperature** – the sampling temperature to use (floating point value between 0.0 and 1.0, default is 0.0).
  - **max_tokens** – maximum number of tokens to use for any individual request.
  - **class_name** – only needed for custom models. Python class used to access the model (typically a LangChain class).
- **vectorSize** (Integer) – the size of the vector produced by an embedding model.
- **distanceFunction** (String) – the distance function used to compare vectors produced by the model.
- **systemPrompt** (Object) – A prompt that will go at the beginning of any prompts submitted to the LLM, whether through SubmitPrompt, AnswerQuestion, or the LLM GenAI Component. The expected format is `{"type": "system", "content": <the prompt>}`.

Custom Operations

SubmitPrompt

Submits a user supplied prompt to the target LLM. Accepts the following parameters:

- *llmName* (String) – the name of the *generative* LLM used to process the submitted prompt.
- *prompt* – the prompt to process. This must be either a *String* value or an array of `io.vantiq.ChatMessage` instances (see ConversationMemory).
- *conversationId* (String) – the optional id of a conversation used to provide additional context for the submitted prompt (see ConversationMemory).

Access Rights

- *admin* – admin users have full access rights to all LLM instances in their namespace.
- *user* – users have read-only access rights to all LLM instances in their namespace.

Logs

Logs provide access to any messages that have been logged by rules executing in the current namespace. A log message resource has the following JSON representation:

```
{
    "invocationId": "4de70a07-043e-43da-a407-8a9266692282",
    "level": "WARN",
    "message": "First Test",
    "sequenceId": 0,
    "timestamp": "2016-05-25T06:04:48.112Z"
}
```

The `invocationId` can be used to group messages produced by the same rule invocation. The `sequenceId` is used to provide an exact ordering of the messages.

Access Rights

- *admin* – admin users have full access rights to all Log instances in their namespace.
- *user* – users have full access rights to all Log instances that they have created.

Namespaces

Namespaces define an isolated environment for a user of the Vantiq Services. Each namespace guarantees complete separation of the data, situations, recommendations and rules from those of all other namespaces. Namespaces may optionally be associated with an organization for purposes of billing and quota management.

The namespace is established at the time the user logs in based on the users identity. Each authenticated identity is associated with one and only one namespace. To create a namespace POST the following JSON document to resource URI:

```
{
  "namespace": "<namespaceName>",
  "organization": {
    name: "<organizationName>",
    description: "<organizationDescription>"
  },
  "username": "<adminUsername>",
  "password": "<adminUserPassword>",
  "encryptionKey": "<encryptionKey>"
}
```

where

- The **namespace** property consists of alphanumeric characters and may contain embedded underscores(_).
- The **organization** is an optional organization instance that should be created and associated with the new namespace.
- The **username** and **password** are both optional with a default value of `admin__<namespaceName>`.
- The **encryptionKey** is optional and if provided is used to encrypt any sensitive data stored in the namespace.

Once created the namespace resource has the following JSON representation:

```
{
  "namespace": "<namespaceName>",
  "organizationRef": "<organizationRef>"
}
```

Custom Operations

Namespaces support the following custom resource operations.

List Authorized Users

- `op` – “getAuthorizedUsers”

Returns a list of the users who are authorized in the namespace, along with the profiles that they have been granted. The operation has no arguments. See example here.

Authorize User

Creates a delegated request which, when executed will grant the executing user access to the namespace. The argument to this operation is a JSON document with the following properties:

- `op` – "delegateUserAuthZ"
- `data` – a JSON document with the following properties:
  - `profiles` – a list of profiles to grant to the user. Optional – if not specified user is granted minimum privilege.
  - `requireExisting` – a boolean which can be set to `true` to indicate that the user being authorized must already be a known Vantiq user. Optional – if not specified then new users are allowed as long the requester is permitted to create them.
  - `parameters` – an object whose key-value pairs will be used to fill out the email template. Optional – the default templates do not require parameters, and they can be set on the publish operation as well. More information on parameters can be found here.
  - `htmlTemplate` – a string that is the name of the Document to use as a template. Can only find Documents in the current namespace. Optional – if not set then it will use the default template, or the template specified in the publish operation. More information on template formatting can be found here.
  - `subject` – a string that will be the subject of emails sent by the request. Defaults to "Vantiq – Namespace Authorization" and may be overridden when publishing the delegated request.
  - `expiresAt` – A DateTime which indicates when the invitation will expire. Optional. Defaults to 2 days from the time the invitation is created. Will be overridden by the *timeout* parameter.
  - `timeout` – The time in seconds that the invitation should last. Optional. Defaults to 2 days or the date in the *expiresAt* parameter.

Revoke Authorization

Revokes authorization for a user to the namespace. This will return a success while doing nothing if you do not have sufficient permissions. The argument to this command is a JSON document with the following properties:

- `op` – "revokeUserAuthZ"
- `data` – a JSON document with the following properties:
  - `username` – the `username` property of the user to revoke.
  - `transferResources` – Optional boolean. Whether to transfer certain resources of the revoked user. If set to true and the revoke succeeds, all resources owned by the revoked user will be transferred to the revoking user.

Transferred Resources

When transferring resources, the below resources will be affected in the specified ways.

- Non-personal tokens. Tokens are typically deleted upon revocation, but will instead have their ownership transferred to the target user. Personal tokens will still be removed, as they use the original creator's username and may have the original creator's permissions in other namespaces.
  - The tokens transferred will be any with `ars_createdBy` equal to the revoked user and will have their `ars_createdBy` field set to the new owner.
- Sources. The permissions of a source are taken from its creator. To avoid them becoming non-functional on revocation, these resources will be transferred.
- The sources transferred will be any with `ars_createdBy` equal to the revoked user and will have their `ars_createdBy` field set to the new owner.
- Groups. Only the owner of a group is allowed to edit the group. To avoid needing to recreate a group when its owner is kicked out, the group's ownership will be transferred.
  - The groups with `owner` equal to the revoked user will have their `owner` and `ars_createdBy` fields changed to the new owner.
- Rules and Apps. These resources use the permissions of the last user to edit the resource. To avoid them becoming non-functional on revocation, these resources will be transferred.
  - Rules and Apps with either the revoked user in `ars_modifiedBy` or the revoked user in `ars_createdBy` while `ars_modifiedBy` is unset, will have `ars_modifiedBy` changed to be the new owner. Rules will be regenerated at this time. Apps will not be fully regenerated, but the underlying Rules and Event Streams will be regenerated.
- Custom Types. Custom types can be made write-by-owner or read-by-owner. For resources that are limited to write-by-owner for admins, all instances of that type created by the revoked user will be transferred.
  - Instances of any write-by-owner or read-by-owner custom types with the revoked user in `ars_createdBy` will have `ars_createdBy` changed to the new owner.

Find Orphaned Resource Owners

Removing users from a namespace can leave non-functional resources behind. This operation returns the owners of any orphaned resources in the namespace, for use with the transfer orphaned resources operation.

- `op` – "findOrphanOwners"

The results will be an array of the usernames for all orphaned resources in the namespaces. These will be the true usernames rather than the preferred usernames, so they may not be human-readable.

Transfer Orphaned Resources

This will transfer all resources orphaned by a specific user to the caller. See the list of transferred resources to see what exactly will be transferred. Note that any tokens will have already been deleted, so transferring will not give the caller any new tokens.

- `op` – "transferResourceOwnership"
- `data` – a JSON document with the following properties
  - `existingOwner` – The username of the current owner of the target resources. If the user still has permissions in the namespace, an error will occur. The find orphaned owners operation should be used to identify valid targets.

Access Rights

- *system admin* – the system admin has full access rights to all namespaces and can create new **organizations** as part of namespace creation.
- *organization admin* – the organization admin has the same rights as a standard namespace admin and can additionally create new namespaces which will automatically be part of their organization.
- *developer* – developers have the same rights as a standard namespace admin and can additionally create new namespaces with themselves as the administrator.
- *admin* – admin users have read and update rights to the namespace instance. Namespaces may only be deleted by the system admin.
- *user* – users have read-only access rights to the namespace instance.

Events

Events are generated for the following operations:

- **authorized** – occurs when a user is authorized for a namespace. The data provided is an object with the properties:
  - *username* – the user's `username` property.
  - *preferredUsername* – the user's `preferredUsername` property.
  - *profiles* – the qualified names of the profiles authorized.
- **revoked** – occurs when a user has their authorization for a namespace revoked. The data provided is an object with the properties:
  - *username* – the user's `username` property.
  - *preferredUsername* – the user's `preferredUsername` property.
  - *profiles* – the qualified names of the profiles authorized.

Examples

Create Namespace with default admin user and password

```
POST https://dev.vantiq.com/api/v1/resources/namespaces
{
  "namespace": "exampleNamespace"
}
```

Create Namespace with default admin user and password and create new organization

```
POST https://dev.vantiq.com/api/v1/resources/namespaces
{
  "namespace": "exampleNamespace",
  "organization": {
    "name": "MyOrganization"
    "description": "The description of my organization."
  }
}
```

Delete Namespace

A namespace is deleted using a DELETE REST request:

```
DELETE https://dev.vantiq.com/api/v1/resources/namespaces/exampleNamespace
```

This operation can only be performed by a namespace admin and it will remove all resource instances associated with the

Nodes

A node identifies a remote Vantiq installation that has agreed to federate with the installation in which the node is defined. A node agrees to federate with a remote node when an administrator authorized to access both the local node and the remote node, defines a node that identifies the remote installation and includes valid credentials to access the remote installation.

Register

A node is registered or updated by issuing a POST on the nodes resource type.

Nodes have the following properties:

- **name** – the local name assigned to the remote node.
- **uri** – the URI(URL) used to access the remote installation. For example, to access the Vantiq public developer installation the URL would be assigned the value: **https://dev.vantiq.com**. In addition to `http(s)` it is also possible to use `ws(s)` in order to use web sockets as the transport or `vqs` to reuse a named web socket connection.
- **username** – the username used to access the remote installation. The username must be a valid username in the remote installation.
- **password** – the password used to authenticate the username.
- **accessToken** – the token used to access the remote installation. Either username/password or accessToken must be specified.
- **credentialType** – set it to the value **userpass** or **token**. Defaults to **userpass** if not specified.
- **deliveryMode** – currently unused but set it to the value **bestEffort**.

- **properties** – a JSON object that identifies properties that can be used to characterize the remote node. The properties are generally used to identify classes of remote nodes when targeting remote invocation in VAIL. See documentation on Selecting Remote Targets.
- **clientOptions** – the JSON object form of Vert.x HTTP Client Options used to configured the client used when connecting to the node.
- **reconnect** – a boolean value indicating whether the node should automatically reconnect if the connection is lost. This property only applies to nodes defining a `ws(s)` URI with the `vqs` option. Defaults to `false` if not specified.

Common uses of `clientOptions` are:

```
Disable the SSL trust check:

{
   "trustAll" : true
}
```

```
Enable the use of an HTTP proxy:

{
   "proxyOptions" :    {
      "host": "hostABC",
      "port": 8080,
      "password": "pwd123",
      "username": "uuuu"
    }
}
```

Named WebSocket connection

If a Node uses web sockets as the transport, it is possible to name the web socket connection using the optional `vqs` parameter: `ws(s)://<host>?vqs=<name>`. Naming a web socket connection allows the remote installation to communicate back through the same open socket using the URI: `vqs://<name>`.

For example, let's assume that a firewall prevents the cloud installation `dev.vantiq.com` from directly accessing an edge server while the edge server can connect to the cloud installation. To allow communication from cloud to edge, we can create a Node definition on the edge (e.g., named EdgeToCloud) with the URI `wss://dev.vantiq.com?vqs=myEdge` and a Node definition on the cloud installation (e.g., named CloudToEdge) with the URI `vqs://myEdge`. As the edge connects to the cloud with the URI `wss://dev.vantiq.com?vqs=myEdge`, the name `myEdge` gets transmitted to `dev.vantiq.com` allowing the cloud to name the open socket. As the CloudToEdge Node definition gets used, the URI `vqs://myEdge` gets resolved to the existing named open socket connection providing a communication channel to the edge.

While a Node with the URI syntax `vqs://<name>` can be defined at any point in time, the Node does require a named open socket connection of the specified `<name>` to resolve to an active connection. For example, as long as the Node EdgeToCloud does not establish a connection, the Node CloudToEdge won't be able to communicate to the edge server.

If you need to have a `vqs` connection always active, specify the `reconnect` property as true in the Node defining the URI `ws(s)://<host>?vqs=<name>`. This will cause the Node to automatically connect as it gets defined and added and also automatically reconnect whenever the connection is lost. To use the above example, the `reconnect` option would be specified on the EdgeToCloud Node definition and it would ensure that EdgeToCloud always keeps the connection open, allowing the CloudToEdge Node `vqs` referenced named socket to always be active.

Note that several Nodes on the same installation can be defined using the same named socket (i.e., using the same URI `vqs://<name>`).

Access Rights

- *admin* – admin users have full access rights to all Node instances in their namespace.
- *user* – users have read-only access rights to all Node instances in their namespace.

Examples

Register Node

```
POST https://dev.vantiq.com/api/v1/resources/nodes
{
    "name": "exampleNodeName",
    "uri": "https://api.vantiq.com",
    "username": "myRemoteUsername",
    "password": "myRemotePassword",
    "deliveryMode: "bestEffort",
    "properties": {
        location: "CA",
        nodeType: "gateway"
    }
 }
```

Delete Node

```
DELETE https://dev.vantiq.com/api/v1/resources/nodes/exampleNodeName
```

Organizations

An organization represents a tenant to whom the system administrator has delegated the authority to create new namespaces. In the case of the Vantiq cloud deployment this is used to represent customers with whom Vantiq has established some form of relationship.

Organizations are created as part of creating their root namespace (see Namespace for details). They have the following properties:

- **name** – the primary id of the organization, consists of alphanumeric characters and may contain embedded underscores(_).
- **description** – an optional description of the organization.
- **k8sResourceUsage** – an optional property containing the organization's current resource usage in the K8s Cluster named `self` (the cluster in which Vantiq runs). This property is present only if the organization has quota to deploy into the `self` cluster and has made use of it.

Access Rights

All users except for the system admin have read-only access to the organization with which their namespace is associated (if any).

Procedures

Procedures are named operations that may be invoked from the body of a rule or another procedure.

Procedures can also be invoked directly via the REST API by issuing a POST operation on the procedure resource instance. The body of the POST contains a JSON document describing the parameters to pass to the procedure. The result of the procedure is returned as the body of the request. Parameters may be passed by name or by position. Note that a mix of named and positional parameters are not supported. For example, the request below would invoke the procedure *ConvertToCelsius* with the parameter *temperature* set to 32.

```
Method: POST
URL: /api/v1/resources/procedures/ConvertToCelcius
Body: { "temperature": 32 }
```

Similarly, to pass the parameter positionally:

```
Method: POST
URL: /api/v1/resources/procedures/ConvertToCelsius
Body: [32]
```

Procedures are identified by their fully qualified name which is a combination of their simple name and the name of the service that they are a part of, if any. For example, the following request invokes the procedure named *ConvertToCelsius* which is part of the *MetricConversions* service.

```
Method: POST
URL: /api/v1/resources/procedures/MetricConversions.ConvertToCelcius
Body: { "temperature": 32 }
```

User defined procedures are described in the VAIL Reference Guide.

Custom Operations

Procedures support the following custom resource operations.

EXECUTE

This operation executes the procedure and returns the result. This operation accepts a single argument which can either be a JSON array or object. If an array is provided then the procedure is invoked using positional parameters where each element in the array is mapped to a procedure parameter in order. If an object is provided then the procedure is invoked using named parameters based on the property names in the supplied JSON object.

Relationships

Procedures may have any of the following relationships:

- *publishes* relationship to any topics that are published to within the Procedure text
- *executes* relationship to any Procedure executed within the Procedure
- *dependency* relationship to any Type, Source, or Service referenced within the Procedure text or parameters
- *dependency* relationship to any Unit Test or Unit Test Suite that tests the Procedure

Access Rights

- *admin* – admin users have full access rights to all Procedure instances in their namespace and read-only rights to any system defined procedures.
- *user* – users have read-only access rights to all Procedure instances.

Group Enabled

Topics are a "group enabled" resource. Placing a "topic" into a group restricts use of the topic (for both publish and subscribe) to identities which are members of the group.

Profiles

A profile contains capabilities with which the user assigned the profile can access data and behaviors defined in the namespace. If a user attempts an operation that is not authorized by their assigned profile, the request will fail returning an authorization error.

Capabilities are defined in the profile properties as follows:

- **dml** – the user can issue select, insert, update, delete and aggregate operations on the collections named in the capability.
- **invoke** – the user can invoke the operations named in the capability.

The profile contains the following properties:

- **name** – the name of the profile. Profile names must be unique. A profile name is a alphanumeric string which may contain embedded underscore characters. No other characters are allowed in a profile name.
- **dml** – defines the capabilities assigned to the user for one or more types defined in the namespace. If capabilities for a specific type are not included, the user has no access capabilities for that type. See DML Capabilities below for additional detail.
- **invoke** – unused except by Vantiq defined types.

The JSON representation of a profile:

```
{
  "name": "<profileName>",
  "dml": { "<typeName">:
      {
        "select": {<typeConstraint>},
        "insert": {<typeConstraint>},
        "update": {<typeConstraint>},
        "delete": {<typeConstraint>},
        "invoke": []
      }
  },
  "invoke": {"createType": true}
}
```

Default Profiles

When a namespace is created the system will automatically create two default profiles named `user` and `admin`. These profiles are immutable and are updated automatically as new types are defined in the namespace.

The specific access level assigned in each profile is determined by the definition of the type. The default is that `user` will be assigned *read access* and `admin` will be assigned *read/write access*. If needed, the access level can be changed for an individual type by setting the properties *userAccessLevel* or *adminAccessLevel* in that type.

Possible System Permission Levels

The admin profile for system types - `admin__system` - can have several different levels of permissions depending on the namespace. When a resource talks about *admin* access levels, it is for all of the following unless otherwise specified.

- **System administrator** – Only exists in the system namespace. Can create organizations and new resources available to all namespaces. Uses the `system.admin` profile.
- **Organization administrator** – Only exists in the initial organization namespace for each organization. Can create new users and a variety of different namespaces. Uses the `system.federatedOrgAdmin` profile.
- **Namespace administrator** – The level for most deployed namespaces. Can create new users in their namespace but cannot create namespaces. Uses the `system.federatedAdmin` profile.
- **Developer** – The level for namespaces created by developers. Cannot create users, though can authorize users into their namespaces and create new namespaces. Uses the `system.federatedDeveloper` profile.

The User Administrator profile for system types - `userAdmin__system` - always has the same level of permissions. It uses the `system.federatedUserAdmin` profile, and only exists in namespaces with the Namespace Administrator. Unless otherwise specified, it has the same permissions as the user profile.

The user profile for system types - `user__system` - always has the same level of permissions. It uses the `system.federatedUser` profile.

There is also the Organization User profile `orgUser__system` in organization namespaces. It uses the `system.federatedOrgUser` profile and gives user-level permissions–plus full read permissions on documents–in all namespaces in the org where the user doesn't have explicit permissions.

DML Capabilities

The dml property value is an object with a property for each type the user is authorized to access. The name of the property is the name of the type. The value associated with the property is an object describing the detailed capabilities assigned to the user for that type using the properties:

- **select** – the value is the constraint(s) on the objects that can be read or null if all objects can be read. For example, a user is only allowed to see financial transactions with values < $100 ('ars_substitution_' is defined later in this document):

`{"select": {"total": {"ars_substitution_lt": "USD:100"}}}`

- **update** – the value is the constraint(s) on the objects that can be updated or null if all objects can be updated. For example, a user is only allowed to update financial transactions with values > $100 (‘ars_substitution_’ is defined later in this document):

  `{"update": {"total": {"ars_substitution_gt": "USD:100"}}}`

- **insert** – the value is the constraint(s) on the objects that can be inserted or null if all objects can be inserted. For example, a user is only allowed to update financial transactions with values > $100 (‘ars_substitution_’ is defined later in this document):

  `{"insert": {"total": {"ars_substitution_gt": "USD:100"}}}`

- **delete** – the value is an object containing the constraint on the objects that can be deleted or null if all objects can be deleted. For example, the user can delete objects only if the total value of the instance is less than $100:

  `{"delete": {"qual": {"total": {"ars_substitution_lt": "USD:100"}}}}`

- **aggregate** – the value is an object containing the constraint(s) on the objects that can be aggregated. For example, The user can aggregate the total property of financial transactions with a total > $100.

  `{"aggregate": {"total": {"ars_substitution_lt": "USD:100"}}}`

If a type is not named in the dml property, the user had no access rights for that type.

Access Rights

- *admin* – admin users have full access rights to all Profile instances in their namespace and read-only access to all system profiles.
- *user* – users have read-only access rights to all Profile instances.

Examples

Create Profile

```
https://dev.vantiq.com/api/v1/resources/profiles
{ "name": "myProfile",
   "dml": {"myType": {
          "select": {},
          "insert": {},
          "update": {"total" : {"ars_substitution_lt": "USD:100"}},
          "delete": {},
          "aggregate":{},
          "invoke": []
},
"invoke": ["createType", "deleteType"]
}
```

Delete Profile

```
DELETE https://dev.vantiq.com/api/v1/resources/profile/myProfile
```

Projects

A Vantiq Project is an collection of resources in the namespace that represent a single deployable unit. Projects are used as part of the Deployment Tool as well as for Integration Tests.

Projects contain the following properties:

- **name** (*String*): the unique name for the assembly
- **resources** (*Array of Objects*): An array of objects each of the form `{resourceReference: </resource>/<resourceId>}` defining each of the resources contained by the Assembly.

Assemblies are special instances of Projects which are configurable and reusable and may be shared via the Vantiq Catalog. Assemblies contain the following additional properites:

- **isAssembly** (*boolean*): a boolean flag that must be set to true if the project is an Assembly
- **configurationProperties** (*Object*): an object containing the configuration properties, where each key is the name of the configuration property and the value is the property description. See below for how to define Configuration Properties
- **configurationMappings** (*Object*): an object describing how each configuration property is applied to the Assembly resources. See the Assembly Guide for how to define Configuration Properties mappings or *usage*.
- **visibleResources** (*Array of Strings*): a list of resource references describing which of the Assembly resources are visible to Assembly Consumers. The Services in this list are considered the Assembly Interface. All other resources listed are simply “Visible Resources”.
- **selectData** (*Object*): an object describing the data to include in the Assembly where each key is a Type name and the value is the “where clause” to use for the SELECT statement.
- **customGenerationProcedure** (*Procedure Name*): the name of a procedure that will handle the assembly configuration and generate the resource configurations. The procedure is provided the parameter: `assemblyName`, which contains the name of the assembly whose configuration is being

generated. If no value is provided, the default procedure will be used.

- WARNING: This procedure *must* create the resource configurations for each resource. It is ***strongly advised*** you use the UI to create the procedure. See here for details

Custom Operations

Relationships

Projects may have any of the following relationships:

- *contains* relationship to project resource contained by the Project
- *dependency* relationship to any Unit Test or Unit Test Suite that tests the Project
- *dependency* relationship to the assemblyconfig used to configure the Project if the Project is an Assembly

Access Rights

- *admin* – admin users have full access rights to all project instances in their namespace.
- *user* – users have no access rights to project instances.

Rules

Rules provide a way for an application to respond to the occurrence of events related to the application model. These events might indicate the arrival of data from an external system, the receipt of data from another application, or a change to the application's persistent data.

See the VAIL Reference Guide for a detailed description of a rule definition.

Register

A new or updated rule or component may be submitted using the **register** command:

```
POST https://dev.vantiq.com/api/v1/resources/rules[?active=false]
<rule definition>
```

If **active** is set to false, the rule is initially placed in the inactive state. By default, new or updated rule definitions are placed in the active state.

See VAIL Reference Guide for a detailed description of a rule definition.

Delete

A rule or component may be deleted by issuing the **delete** command:

```
DELETE https://dev.vantiq.com/api/v1/resources/rules/<ruleName>
```

Delete the named rule or component.

Change Rule Activation State

A rule or component can be set to the active state via a REST request:

```
PATCH https://dev.vantiq.com/api/v1/resources/rules/<ruleName>
```

with the JSON message body:

```
[
    {
        "op": "replace",
        "path": "/active",
        "value": true
    }
]
```

A rule can be set to the inactive state via a REST request with the JSON body:

```
[
    {
        "op": "replace",
        "path": "/active",
        "value": false
    }
]
```

Relationships

Rules may have any of the following relationships:

- *subscribes_to* relationship to the resource that produces the Rule's triggering event(s)
- *publishes* relationship to any topics that are published to within the Rule text
- *executes* relationship to any Procedure executed within the Rule
- *dependency* relationship to any Type, Source, or Service referenced within the Rule text
- *dependency* relationship to any Unit Test or Unit Test Suite that tests the Rule

Access Rights

- *admin* – admin users have full access rights to all Rule instances in their namespace and read-only access to all system rules.
- *user* – users have read-only access rights to all Rule instances.

Scheduled Events

Scheduled Events are events which have been scheduled for delivery at some later point in time. The schedule may specify that they be delivered once or repeatedly until deleted. Such events can be created via the resource API or by using the PUBLISH statement in VAIL.

Scheduled Events have the following properties:

- **name** – the name of the scheduled event. This is a read-only property which serves as the resource identifier. It must be provided when creating a periodic event, but can be left off of requests to create a one-time event (which will result in a system generated name).
- **periodic** – a Boolean which indicates whether this is a one-time or periodic event. This value is immutable.
- **isTransient** – a Boolean which indicates whether this is a "transient" (in-memory only) scheduled event.
- **occursAt** – a DateTime which indicates when the next delivery of the event is scheduled. For one-time events this will be the only time the event is delivered.
- **interval** – an Integer which indicates how frequently an event should be delivered. The value is given in milliseconds. Can be used when creating either one-time or periodic events, but will only be returned for periodic events.
- **topic** – the topic on which the event will be published.
- **message** – the message data to publish.

Once created, one-time events can only be deleted, they may not be updated.

Custom Operations

Relationships

Scheduled Events have a *publishes* relationship to the Topic that it is scheduled to publish to.

Access Rights

- *admin* – admin users have full access rights to all Scheduled Event instances in their namespace.
- *user* – users have read-only access rights to all Scheduled Events instances in their namespace.

Examples

Create a one-time Scheduled Event

Here we create a one-time event that will be delivered 1 minute after our request:

```
POST https://dev.vantiq.com/api/v1/resources/scheduledevents
{
  "interval": 60000,
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

```
{
  "name": "12345678",
  "periodic": false,
  "occursAt": "<ISO date string one minute from request time>",
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

Here we create a one-time event that will be delivered at the specified time (which must be in the future):

```
POST https://dev.vantiq.com/api/v1/resources/scheduledevents
{
  "occursAt": "<ISO date string specifying a time in the future>",
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

```
{
  "name": "9081399291",
  "periodic": false,
  "occursAt": "<ISO date string specifying a time in the future>",
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

Create a periodic Scheduled Event

Here we create a scheduled event that will be delivered one hour from the request and every subsequent hour.

```
POST https://dev.vantiq.com/api/v1/resources/scheduledevents
{
  "name": "myEvent",
  "periodic": true,
  "occursAt": "<ISO date string specifying a time in the future>",
  "interval": 3600000,
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

```
{
  "name": "myEvent",
  "periodic": true,
  "occursAt": "<ISO date string specifying a time in the future>",
  "interval": 3600000,
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

Get a Scheduled Event definition

```
GET https://dev.vantiq.com/api/v1/resources/scheduledevents/myEvent
```

```
{
  "name": "myEvent",
  "periodic": true,
  "occursAt": "<ISO date string specifying a time in the future>",
  "interval": 3600000,
  "topic": "/myTopic",
  "message": { "value": "This is a string" }
}
```

Update Scheduled Event interval

```
PUT https://dev.vantiq.com/api/v1/resources/scheduledevents/myEvent
{
  "name": "myEvent",
  "interval": 60000
}
```

Secrets

Secrets are storage containers for secure text that can be written once, and then never seen again by users in the namespace. Secrets can be used to configure certain properties in source configurations, which adds a layer of security to the source by hiding the secret value to users viewing the source configuration.

Secrets have 3 properties (all strings): name, description, and secret. When secrets are selected from the database, the secret property is not returned, and the secret value can only be utilized by sources that reference the secret by name. The secret property is encrypted in the database as an extra layer of security.

The source configuration properties that can reference secrets include:

- MQTT – password
- AMQP – password
- REMOTE – password and accessToken
- CHATBOT – clientSecret
- EMAIL – password
- PUSH_NOTIF – aPNSPKCS12Password and firebaseServerKey
- SMS – twilioAuthToken

Examples

Create a Secret

```
POST https://dev.vantiq.com/api/v1/resources/secrets
```

```
{
  "name": "MySecret",
  "description": "This is a description of my secret.",
  "secret": "mypassword"
}
```

Fetch a Secret

```
GET https://dev.vantiq.com/api/v1/resources/secrets/MySecret
{
  "name": "MySecret",
  "description": "This is a description of my secret."
}
```

Notice that when performing a GET on a secret, the secret property is omitted from the response. The secret value can be overwritten with an update, but can never be selected by a user request.

Updating Secrets

While secret values cannot be seen after they are initially set, the secret value can be updated. One note about updating secret values is that any source that references the updated secret must be deactivated and reactivated to have the secret change take effect. Until the source is reactivated, the source will continue using the old secret value.

Semantic Indexes

Semantic Indexes support semantic search over specific content contained in each index instance.
See also built-in service io.vantiq.ai.SemanticSearch.

Semantic Indexes have the following properties:

- **name** (String) – the name of the semantic index. Must be unique in a given Vantiq namespace. May include a package name.
- **embeddingModel** (String) – the name of the LLM used to create vector embeddings for any content. The LLM type must be *embedding*. The property is immutable once set.
- **description** (String) – an optional description of the index.
- **config** (Object) – Configuration for the semantic index. The configurable properties are:
  - **rephraseQuestion** (Boolean) – whether to use conversation history and other context to rephrase the question provided. Defaults to `true`.
  - **returnGeneratedQuestion** (Boolean) – whether to include the rephrased question in the returned object when answering a question. Note that if **rephraseQuestion** is false, the rephrased question will be returned but the original question will be used. Defaults to `false`.
  - **returnSourceDocuments** (Boolean) – whether to return the metadata on the documents included as context for the question. Defaults to `true`.
- **defaultMinSimilarity** (Real) – the default minimum similarity score required for a document in the semantic index to be considered relevant in answering a question. When no value is provided, the semantic index will always return the most relevant documents available regardless of similarity score.
- **externalProvider** (String) – an optional function name used by the GenAIFlow service connector to establish a connection to an external semantic index engine.
- **remoteTarget** (String) – an optional parameter used to identify the target of a remote semantic index. See here for more details on remote semantic indexes. When setting it directly, the format is `<target index name>:<node to target index's namespace>`. The token used for the node is expected to be user-level.
- **databaseConfig** (Object) – an optional configuration document used to configure the index's vector DB. This is currently used when configuring access to an *external* semantic index provider.
- **qaProcedure** (Resource Reference) – an optional reference to a procedure that can be used to answer questions against the index. The procedure accepts a String and a runtime configuration. This specification is required to facilitate the use of an external semantic index with AnswerQuestion activity patterns. It cannot be used with internally managed semantic indexes.
- **ingestProcedure** (Resource Reference) – an optional reference to a GenAI Flow procedure that can be used to ingest content into the index. The procedure accepts a String and a runtime configuration. This specification enables users to alter the default ingest mechanism and customize the steps taken. Further, ingest via GenAI Flow procedures relaxes the time constraints on the ingest process, allowing the utilization of more complex and long-running operations.
- **entries** (Array of ArsSemanticIndexEntry) – a read-only list of the entries that have been loaded into the index. Only included when explicitly requested. To avoid retrieving all semantic index entries, which may be a significant number on larger indexes, you should instead perform selects on the ArsSemanticIndexEnty resource.
- **defaultQAModel** (String) – the optional name of a *generative* LLM used when answering questions against the index.

Custom Operations

Add Entries

Accepts an Array of `ArsSemanticIndexEntry` instances describing content to be loaded into the index. The expected properties are:

- **id** (String) – the entry's id. This value is optional. If not provided, then an id will be generated for the new entry. If a value is provided and it refers to an existing entry, then the entry will be "updated" using the new content.
- **contentType** (String) – the optional MIME type of the content.
- **fileName** (String) – the optional name of the file containing the content.
- **metadata** (Object) – metadata associated with the entry.

The following properties are all optional, but one of them must be provided. The first one found (in the order given below) will be used:

- **content** (String) – the content to be loaded. If the content type is binary, then it is assumed that this value will be Base64 encoded.
- **resource** (ResourceReference) – the Vantiq resource from which to read the content.
- **uri** (String) – the URI from which to read the content.

Remove Entries

Accepts an Array of String values, each of which is assumed to be the id of an existing entry (any values that don't match will be ignored). The referenced entries will be removed from the index.

Clear Entries

Removes all entries from the index.

Answer Question

Uses a generative LLM to synthesize an answer to a submitted question using context obtained from the target semantic index.

Accepts the following parameters:

- *question* (String) – the question to answer using the context provided by the index.
- *qaLLM* (String) – the optional name of a generative LLM to use to produce the answer. If no value is provided, then the index's default Q&A LLM will be used (assuming it has been set).
- *conversationId* (String) – the optional id of a conversation used to provide additional context for the question being asked. See ConversationMemory.
- *minSimilarity* (Real) – an optionally specified number between 0 and 1 denoting the minimum similarity score required for a document in the semantic index to be considered relevant in answering the question. If unspecified, the default is taken from the semantic index definition. If no value is specified in the definition, answer question proceeds without considering a minimum score threshold using the most similar documents. Should the minSimilarity threshold result in no relevant documents from the semantic index, answer question returns "I don't know" without consulting the qaLLM.
- *contextPrompt* (String) – an optional parameter that can prompt the LLM to focus on particular context within the ongoing conversation that is most relevant to answering the follow-up question.

The result is an Object value with the following properties:

- *answer* (String) –the answer to the submitted question.
- *metadata* (Object) – the metadata associated with the entry(ies) used to provide context when answering the question. Excluded if **returnSourceDocuments** is set to `false`.
- *rephrasedQuestion* (String) – the question rephrased by the LLM using the relevant context. If **rephraseQuestion** is set to `false` this will still contain the rephrased question, but it will not be the question asked to the LLM. Included only if **returnGeneratedQuestion** is set to `true`.

Retrieve Chunks

Retrieves the contents of the specified entry in the target semantic index.

Accepts the following parameters:

- *indexName* (String) - the name of the SemanticIndex to retrieve the content from.
- *entryId* (String) - the id of the entry whose content will be retrieved.
- *limit* (Integer) - the maximum number of chunks to retrieve at a time.
- *offset* (String) - the id of the chunk to start retrieving from. Starts at the first chunk if `null`. This should be taken from the *offset* field of the returned Object.

The result is an Object value with the following properties:

- *offset* (String) – the offset to be used in future *retrieveChunks()* calls to continue where this call left off. `null` if *points* contains the last chunk of this entry.
- *points* (Object Array) - the points stored in the database. They will be in the following format:
  - *content* (String) - the content of this chunk.
  - *metadata* (Object) - the metadata for this chunk.

Restore Index From DB

Update the definition of an index using the current state of the vector DB. Note that the system does not try to completely "lock" the index during this operation, so it may produce an inaccurate result if changes are being made while it is run.

Relationships

Semantic Indexes may contain the following relationships:

- *dependency* relationship on the embedding and (if set) default Q&A LLMs.

Semantic Index Entries

Semantic Index Entries represent the contents of a Semantic Index. They are manipulated using the Semantic Index operations, and cannot be directly interacted with except for being read by admins.

Semantic Index Entries have the following properties:

- **indexName** (String) – the name of the Semantic Index that this entry belongs to.
- **id** (String) – the entry's ID, used when updating and removing entries. Ids are unique *per index* so it's possible to have two entries with the same *id*, as long as they each have a different *indexName*.
- **status** (String) – the current status of the entry. One of the following:
  - *uploadPending* – waiting on a file upload to complete. This will be a TUS upload to the location specified by the *uploadTarget* field.
  - *loading* – the content has been uploaded, and is now being processed and stored in the Semantic Index.
  - *loaded* – the entry has been fully stored in the Semantic Index and is available for use.
  - *failed* – an error occurred while retrieving, processing, or storing the entry.
  - *empty* – no content could be found for the entry.
- **error** (String) – the message for the error that occurred while creating the entry, if any occurred. Only present when the status is *failed*.
- **contentType** (String) – the MIME type of the provided content.
- **metadata** (Object) – the metadata to be associated with the entry. Can be used when filtering Semantic Index searches.

The remaining properties are only relevant to the Add Index Entry operation or internal server operations, and can be ignored when reading entries.

- **fileName** (String) – the name of the file that provided the content. Automatically added to *metadata* when present.
- **uri** (String) – the URI from which the contents were retrieved. Automatically added to *metadata* when present.
- **httpHeaders** (Object) – headers that are used when provided.
- **uploadTarget** (String) – the location to which the entry's contents should be uploaded. This will be a TUS upload. Only appears when no content was provided when adding an entry. Server-managed.
- **content** (String) – the content of the entry. Only present when explicitly included in the Add Entry operation.
- **resource** (String) – the Resource Reference for the Vantiq resource that created the entry. Only present when explicitly included in the Add Entry operation, or when an the entry content was uploaded.
- **fromContent** (Boolean) – indicates whether the content was directly included in the message. Server-managed.
- **contentUri** (String) – the uri from which the content was retrieved. Server-managed.

Access Rights

- *admin* – admin users have read-only access rights to all Semantic Index Entries in their namespace. To edit entries, use the relevant Semantic Index operation on the entry's index.
- *user* – users have no access rights to Semantic Index Entries.

Service Connectors

Service Connectors represent the implementation of a Service outside of the Vantiq Platform using a language other than VAIL. These implementations can either be deployed and managed "externally" (with no support from Vantiq) or managed as a resource in a Kubernetes cluster. That cluster can be an external Kubernetes cluster or the Vantiq Kubernetes cluster. Either way the connector must be reachable over a network using WebSockets and must implement the Vantiq Service Connector protocol.

Service Connectors have the following properties:

- **name** (String) – the name of the service connector. Must be unique in a given Vantiq namespace. May include a package name.
- **isExternal** (Boolean) – indicates whether the connector will be deployed externally or in the Vantiq Kubernetes cluster.

External connectors have the following additional properties:

- **host** (String) – the name of the host on which the service connector is running. The default value is `localhost`.
- **port** (Integer) – the port at which the service connector can be contacted. The default value is `8888`.

Kubernetes based connectors have the following additional properties:

- **image** (String) – the name of a Docker image containing the runtime implementation of the connector. Required when deploying via Kubernetes.
- **vCPU** (Integer) – an optional number of virtual CPUs required by the connector.
- **memory** (Integer) – an optional amount of memory (in KB) required by the connector.
- **secret** (ResourceReference) – a reference to a Vantiq Secret resource which will be mounted as the file `/opt/vantiq/secret/secret.properties` in the service connector's file system.
- **replicaCount** (Integer) – the number of connector instances to start (default is 1).
- **pullSecret** (`io.vantiq.k8s.HybridResourceRef`) – a reference to secret value used as the Kubernetes "pull secret". The reference can be to either a Vantiq Secret or an existing Kubnernetes secret (only valid for service connectors in the system namespace).
- **pullPolicy** (String) – the pull policy to use for the connector's image. Legal values are `IfNotPresent` (default), `Always`, or `Never`.
- **resourceMounts** (`io.vantiq.k8s.ResourceMount` Array) – an optional list of resources that should be mounted into the service connector's file system. Each mount has the following properties:

- **resourceRef** (ResourceReference) – the Vantiq or Kubernetes resource whose content should be mounted.
- **mountPath** (String) – the file system path used to mount the resource.
- **volumeName** (String) – the optional name of the volume to be mounted.
- **isExternal** (Boolean) – if `true` then the resource reference is to a Kubernetes resource. Otherwise (the default) it refers to a Vantiq resource.
- **optional** (Boolean) – indicates whether the service connector will start if the target resource does not exist. The default value is `false`.

Access Rights

- *admin* – admin users have full access rights to all ServiceConnector instances in their namespace.
- *user* – users have read-only access rights to all ServiceConnector instances in their namespace.

Services

Services encapsulate behavior associated with a specific functional aspect of an application. Services expose that behavior via their interface, which consists of a list of procedures and/or Event Types which are available to consumers of the Service. Services may be shared with other applications via the Vantiq Catalog which allows their interface to be accessed remotely.

Services have the following properties:

- **name** – the name of the service. This is a read-only property which serves as the resource identifier.
- **description** – the description of the service
- **globalType** – the name of the type that represents the state for global stateful procedures. Each field in the type will be a stateful variable of the same name in all global procedures of the service.
- **partitionedType** – the name of the type that represents the state for partitioned stateful procedures. Each field in the type will create a stateful variable of the same name in all partitioned procedures of the service.
- **scheduledProcedures** – an object that identifies the scheduled procedures and their intervals. Only valid for stateful services. Each scheduled procedure must be either global or multi-partitioned. If a listed procedure is neither of these (or doesn't exist) then the service cannot run. The format is `{<procedure name>: <interval in ms>, ...}`.
- **eventTypes** – an object that identifies the event types used to describe the production and consumption of events by the owning Service. The keys are the name of the Event Types which must be legal identifiers and unique for a given Service. The values are the Event Type definitions:
  - **description** – the description for the Event Type.
  - **direction** – indicates which way the events associated with the Event Type "flow". Must be one of:
    - **OUTBOUND** – indicates that events flow from the Service to the consumer of the Service. *Outbound* Event Types can be referenced by the `WHEN` clause of a rule or as the triggering condition of an App.
    - **INBOUND** – indicates that events from the consumer to the Service for processing. *Inbound* Event Types can be used as the target of a `PUBLISH` statement.
  - **implementingResource** – The resource, either a Rule or an App that consumes the Inbound Event and implements its behavior.
  - **isReliable** – indicates whether the Service Event Type will be processed "reliably" (see the Reliable Messaging Guide for more details).
  - **eventSchema** – a reference to a Type which describes the structure of the data associated with events of this type.
- **interface** – a list of the interfaces of all public procedures in the service. This is may be explicit set or otherwise dynamically generated for local and system services, and must match the publisher's interface for catalog services. Its *name* and *typedParameters* fields will be validated against the actual procedures when publishing and subscribing. The fields for each Object are below, and typically match fields of the same name in procedures:
  - **name** – the name of the procedure. Must not include the service name.
  - **returnType** – an object identifying the type of the return value. Can be null or unspecified. Format is as follows:
    - **type** – a string that is name of the return type. Must be one of the valid VAIL types or a custom type.
    - **multi** – a boolean stating whether to expect an array as a result.
  - **description** – a string describing the procedure. Can be null or unspecified.
  - **parameters** – a list that provides details for each parameter. It must be in the same order as the actual parameters. The fields for each Object are as follows:
    - **name** – the name of the parameter. This value will always be used in validation.
    - **type** – a string identifying the type of the parameter. Must be one of the valid VAIL types or a custom type. If the actual procedure is typed, then this will be used in validation. Note that the actual parameter's type should be `Object` if the listed type is a custom type.
    - **multi** – a boolean indicating whether the parameter is expected to be an array. This value will always be used in validation.
    - **description** – a description of the parameter. This value is never used in validation. It may be null or unspecified.
    - **required** – a boolean indicating whether the parameter is required by the procedure. This value is never used in validation. It may be null or unspecified.

Standard Operations

EXECUTE

Services support the `execute` operation on their procedures. The resource id for the request is the qualified service procedure name of the form `<serviceName>/<procedureName>`. The body of the request is the parameters to be passed to the procedure. These can be given in either positional form (an array) or named parameters (an object):

```
PUT https://dev.vantiq.com/api/v1/resources/services/<serviceName>/<procedureName>
{
    <procedure parameters>
}
```

PUBLISH

Services support the `publish` operation on their *inbound* event types. The resource id for the request is the qualified service event type of the form `<serviceName>/<eventTypeName>`. The body of the request is the data to be published. For example in the REST binding the request would be:

```
POST https://dev.vantiq.com/api/v1/resources/services/<serviceName>/<eventTypeName>
{
    <eventData>
}
```

Status

In addition to the standard `health` and `metrics` properties, the response from the `status` operation may include:

- `stateInitialized` – `Boolean` value indicating whether or not the service's state has been initialzed. Will only be present for stateful services.
- `scheduledProcedures` – `Object` value containing information for each of the service's scheduled procedures (if any). The names of the `Object` properties correspond to currently known scheduled procedures. Each procedure entry may have the following properties:
  - `interval` – the current scheduling interval for the procedure.
  - `eventName` – the name of the scheduled event used to execute the procedure (if not present then the scheduled procedure is not currently active).

Custom Operations

Publish

An event is published to a Service Inbound Event Type by issuing a publish request on the Event Type name with the body of the request containing the event data associated with the publication.

```
POST https://dev.vantiq.com/api/v1/resources/services/<serviceName>/<eventTypeName>
{
    <eventData>
}
```

Relationships

Services may contain the following relationships:

- *contains* relationship to all of the Procedures within the Service.
- *dependency* relationship on any Types used to contain the Service State or referenced as the event schema on any Service Event Types.

Access Rights

- *admin* – admin users have full access rights to all Service instances in their namespace and read-only access to all system services.
- *user* – users have read-only access rights to all Service instances.

Examples

Get a Service definition

```
GET https://dev.vantiq.com/api/v1/resources/services/MyService
```

```
{
  "name": "MyService",
  "description": "This is a description of my service.",
  "interface": ["firstProcedure", "secondProcedure"]
}
```

Update Service description

```
PUT https://dev.vantiq.com/api/v1/resources/services/MyService
{
  "name": "MyService",
  "description": "This is a new description for my service."
}
```

Sources

A source describes an external system that either sends data to the Vantiq server or receives data and notifications from the Vantiq server.

An overview of external system integration via sources can be found in External Source Reference Guide There are a variety of external sources predefined in Vantiq:

- EMAIL – for sending email to designated users. See EMAIL Source Integration for details.

- SMS – for sending text messages to designated users. See SMS Source Integration for details.
- MQTT – for sending and receiving data using MQTT. See MQTT Source Integration for details.
- AMQP – for sending and receiving data using AMQP. See AMQP Source Integration for details.
- REMOTE – for sending and receiving data using a REST API. See REMOTE Source Integration for details.
- VIDEO – for receiving frames from a video (online camera or file). See VIDEO Source Integration for details.
- CHATBOT – for sending and receiving data from Chatbots built using the Microsoft Azure Bot Services. See CHATBOT Source Integration for details.
- KAFKA – for sending and receiving data using KAFKA. See KAFKA Source Integration for details.
- GCPS – for sending and receiving data using Google Cloud Pub/Sub. See GCPS Source Integration for details.

Additionally, new external source types can be created using the **Enterprise Connector** capability. See Enterprise Connectors Reference Guide for details.

A source is created or updated by issuing a POST on the sources resource type. All sources must define a standard set of properties and an additional, source type specific set of configurations parameters. The JSON form of a source is as follows:

```
{
    "name": "<sourceName>",
    "type": "<sourceType>",
    "config": {
        <source type specific parameters>
    }
    "autoUnwind": <automatically unwind events>
}
```

where:

- **name** – the name assigned to the source.
- **type** – the type of source. Currently, a type specified by an Enterprise Connector or one of these string values:
  - EMAIL
  - SMS
  - MQTT
  - AMQP
  - REMOTE
  - CHATBOT
  - KAFKA
  - GCPS
- **config** – contains the source type specific parameters associated with the source. The detailed contents of **config** is defined in the source type specific documentation referenced above.
- **autoUnwind** – indicates whether the source should automatically “unwind” any event data that it receives. If `true`, then if an array is received as event data, the system will immediately process the array and generate one event for each array member (instead of one containing the array itself).

A source is deleted by issuing a DELETE on the sources resource supplying the name of the source to delete.

Custom Operations

Sources support the following custom resource operations.

PUBLISH

Publishes data to the specified source. This operation accepts one argument, the data to publish. This is expected to be a JSON document.

QUERY

Performs a query operation against the specified source. This operation accepts one argument, a description of the query to perform. This is expected to be a JSON document. The details for which sources support querying and the format of their query document can be found in the documentation for each source type.

DEPLOY

This deploys a connector image for the source. The parameters are identical to those of the K8s Cluster Deploy operation with the addition of the `clusterName` which names the cluster to which to deploy. (See the example below for details.)

The operation is invoked using the source’s resource URI with “/command’ appended.

FETCHLOGS

This fetches the logs for a K8s Installation associated with a connector. The parameters are identical to the K8s Cluster FetchLogs operation with the addition of the `clusterName` which names the cluster containing the installation on which to operate.

RESTART

This restarts a K8s Installation associated with a connector. The parameters are identical to the K8s Cluster Restart operation with the addition of the `clusterName` which names the cluster containing the installation on which to operate.

The operation is invoked using the source’s resource URI with “/command’ appended.

SHUTDOWN

This shuts down a K8s Installation associated with a connector. The parameters are identical to the K8s Cluster Shutdown operation with the addition of the `clusterName` which names the cluster containing the installation on which to operate.

The operation is invoked using the source's resource URI with "/command' appended.

UNDEPLOY

This performs an *undeploy* operation on the K8s Installation connector associated with this source. The parameters are identical to the K8s Cluster Undeploy undeploy operation with the addition of the `clusterName` which names the cluster containing the installation on which to operate.

The operation is invoked using the source's resource URI with "/command' appended.

Relationships

Sources may have any of the following relationships:

- *dependency* relationship to the Type used as the messageType
- *executes* relationship to the Mock Publish or Mock Query Procedures
- *dependency* relationship to the Source Implementation used by an Enterprise Connector

Access Rights

- *admin* – admin users have full access rights to all Source instances in their namespace and read-only access to all system sources.
- *user* – users have read-only access rights to all Source instances.

Group Enabled

Sources are a "group enabled" resource. Placing a "source" into a group restricts use of the source (for both publish and subscribe) to identities which are members of the group.

Examples

Create Source

Create (or update) an MQTT source:

```
POST https://dev.vantiq.com/api/v1/resources/sources
{
    "name": "myMqttSource",
    "type": "MQTT",
    "direction": "SOURCE",
    "config": {
       "serverURIs": ["tcp://me.vantiq.com:1883"],
       "topics": ["com.accessg2.stream.mqtt.example"],
       "keepAliveInterval": 30,
       "connectionTimeout": 30,
       "cleanSession": true,
    }
}
```

Delete Source

A user defined node may be deleted by using the **deleteNode** REST request supplying the name of the node to delete. Example:

```
DELETE https://dev.vantiq.com/api/v1/resources/sources/myMqttSource
```

Deploy

```
POST https://dev.vantiq.com/api/v1/resources/sources/extSource/command
{
    "op": "deploy",
    "data": {
        "clusterName": "clusterForMySource",
        "name": "connectorInstallation",
        "k8sNamespace": "myNamespace",
        "config": [
            {
              "type": "image",
              "name": "myRepo/connectorImage"
            },
            {
              "type" : "file",
              "content" : "some data",
              "path" : "/app/somedirectory",
              "filename": "myfile",
              "contentName": "configMapKey"
            }
          ]
    }
}
```

Note that the `config` data will be image specific.

This operation will return the K8s Workitem for the request created. For the operation performed above, this will look like the following:

```
{
   "requestId": "<some UUID>",
   "clusterName": "clusterForMySource",
   "operation": "deploy",
   "timestamp": <time of request>
   "status": "REQUESTED"
   "request": <config value from request>
   "requestedBy": "<name of the Vantiq user who performed the deploy>"
}
```

StorageManagers

Data retrieval needs for different applications can vary significantly. One application might work heavily with timeseries data, while another focuses on analytics. There is no one storage management solution that works well in all scenarios. Further, applications may need to work with data stored externally to Vantiq where it is not practical to import a copy. The storage manager resource addresses these issues by enabling users to add pluggable storage management capabilities to meet their specific needs. Once a user successfully installs and configures a storage manager, applications can access the external data as though it were stored locally to the Vantiq system. Further, an application may leverage any special capabilities inherent to the remote data storage system with the understanding that operations specific to a particular data store are unlikely to work in other contexts. The trade-off inherent in taking advantage of these specialized features is that applications give up some degree of data store independence.

Properties

In addition to the name there is a single property for a storage manager: *implementingService*. This identifies the service responsible for handling all requests for data associated with the storage manager.

Storage Manager Reference in Type Definition

Type definitions whose underlying data are handled by a storage manager contain a storage manager reference. In addition to naming the storage manager for the external data, the reference may also designate how to connect to the external store and may define additional properties as name, value pairs. This mechanism marries the type system with external storage management enabling applications to work with external data in the same manner as locally stored data subject to the capabilities of the underlying system. See also the storageManager property for the Types resource

Temp Blobs

Temp Blobs are used to hold larger entities (documents, images, videos) in memory within the server. They can be created by a REMOTE or VIDEO source.

Temp Blobs have an identification that is assigned by the Vantiq system. They have a relatively short lifetime (measured in minutes), after which they are destroyed. They are memory-resident only, intended to avoid the expense of saving an image to long-term storage when that image is not likely to need to be saved after analysis. Because they are memory-resident, they are available only within the local Vantiq instance.

Temp Blobs can be copied to a document, image, or video using the Copy mechanism and the `fromTemp` property. See Create Document from Temp Blob for an example using a document. Similar functionality is described for images and videos.

No REST operations on Temp Blobs are supported.

Properties

A Temp Blob contains the following properties visible to the VAIL consumer.

- **fileType** The MIME type of the content
- **contentSize** The size of the entity contained
- **contentRef** A Resource Reference to the Temp Blob

Access Rights

- Users within the namespace in which the Temp Blob was created can read or use the data using services or operations on other types.
- No update operations are supported.

TensorFlowModels

A TensorFlow model (an instance of the `tensorflowmodels` resource) represents a specific type of neural net model stored in the Vantiq system.

TensorFlow Model Parts

YOLO TensorFlow Models

(For more information about YOLO TensorFlow models, please see the Image Processing Reference Guide.)

YOLO TensorFlow models require the uploading of two (2) files: a JSON file known as the meta file that contains the meta information (names of labels, controlling information about the model), and the model itself (a protobuf representation of the model). The meta file is typically hundreds to thousands of bytes; the model file can be hundreds of megabytes.

When creating a YOLO TensorFlow model, use a `modelType` of `tensorflow/yolo`. Note that the *modelType* `tensorflow/yolo` is used for both YOLO v2 and YOLO v3 models. The associated meta file contains sufficient information to distinguish the versions.

Information about generating meta file and the model itself can be found in the Image Processing Reference Guide.

TensorFlow models (non-YOLO)

More general TensorFlow models can include a meta file, but it is not required. When used, only the meta file section describing the input and output operations is used.

When creating a (non-YOLO) TensorFlow model, use a `modelType` of `tensorflow/plain`.

Creating & Updating Tensorflowmodel Instances

TensorFlow models that do not use a meta file are created or updated by sending the JSON representation of the model to the resource URI. The JSON representation of the model is as follows:

```
{
    "name": "<model name>",
    "modelType": "tensorflow/yolo" | "tensorflow/plain",
    "contentSize": <size of model to be uploaded>
}
```

For TensorFlow models that use a meta file, upload the meta file using a multi-part MIME message as follows:

```
POST https://dev.vantiq.com/api/v1/resources/tensorflowmodels?modelName=<name of model>&modelType=<appropriate model type>&modelLen
--<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: application/json
<meta content>
--<your boundary marker>
```

The *modelLength* is used by the system to know when all segments have been uploaded.

Uploading Model Content

In either case, the actual model content should be uploaded using the *tus* protocol. The *tus* protocol provides for segmented, restartable uploads.

Once the model as been created or updated, the instance will exist in the Vantiq system, but it will be marked as *incomplete*. *Incomplete* models cannot be used for analysis until they are made *complete*. When the uploading of segments completes (when sufficient data to satisfy *contentSize* or *modelLength* has been provided), the model will be marked *complete*.

To complete the model, the model content will be uploaded by segment using the *tus* protocol. The description of the *tus* protocol can be found in the *tus* specification. At a high level, the sequence of events is as follows:

- Issue an HTTP OPTIONS request against the TensorFlow model. The result will contain the following headers:
  - `Tus-Max-Size` – Upload size limit for each segment to be uploaded
  - `Tus-Extension` – Will contain information about extensions to the protocol
    - `Vantiq-Patch-Multipart-Form` – Indicates that we support multi-part MIME patch requests. Uploading via multi-part MIME is very strongly encouraged.
    - `Vantiq-Multipart-Max-Size` – Limit for upload size using this extension

    - `Vantiq-Multipart-Max-Size` – Value of 'true' indicates that this is an edge installation
- Issue an HTTP HEAD request against the instance. The result will contain the following headers:
    - `Upload-Length` – Expected size of the complete model. This was included in the multi-part MIME request that created the tensorflowmodel.
    - `Upload-Offset` – The offset at which the next upload should start.
- Upload the data segments
    - Each upload must contain the appropriate *tus* headers:
        - `Tus-Resumable` – should contain `1.0.0`
        - `Upload-Offset` – the offset at which this segment is to be placed. This should match the `Upload-Offset` header returned by the HEAD request above.
        - `Content-Type` – should be set to `multipart/form-data` if using multi-part MIME, `application/offset+octet-stream` otherwise.
    - If multi-part form PATCH requests are supported,
        - PATCH the segment as form data, setting the part's content-type: `application/offset+octet-stream`
    - If multi-part form PATCH requests are not supported (or not available),
        - send a PATCH request containing the segment to the URL.
- Repeat the upload, updating the offset as each segment is uploaded. If desired, you can repeat the HEAD request, but it is not necessary.

Any HEAD request on a specific instance of `tensorflowmodels` will return the current state of the upload process, as per *tus*. If not all segments can be uploaded (*e.g.* some network disconnection occurs), a subsequent HEAD request will let you know where in the process things are. The next upload can, then, resume the process.

Additionally, on an edge installation, TensorFlow model instances may be created by uploading the model and meta file in one (typically, very large) multi-part MIME message. It is not necessary to use this as edge installations support the segment upload described above; it is provided for simplicity.

Uploading a model using a multi-part MIME message is accomplished using a message of the following form:

```
POST https://dev.vantiq.com/api/v1/resources/tensorflowmodels?modelName=sampleModel&modelType=tensorflow/yolo
--<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: application/json
<meta content>
--<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: application/octet
<model content>
--<your boundary marker>--
```

where:

- `<assignedName>` is the name of the TensorFlow model as stored in `tensorflowmodels` instance.
- `<fileName>` is the name of the file from which the model content or meta information was obtained.

In addition, the appropriate HTTP headers must be set:

```
ContentLength: <length of http content in above message>
ContentType: multipart/form-data; boundary=<your boundary marker>
```

Once the instance has been created, the TensorFlow model's information will be available via the resource URI:
`api/v1/resources/tensorflowmodels/<modelName>`.
This will return the following JSON representation:

```
{
    "name": "models/mymodel",
    "modelType": "<model type in question>",
    "metaFile": <some JSON object>,
    "content": "/tfmodels/models/mymodel"
}
```

The `content` property is the absolute URI that can be used to retrieve the file's content from the Vantiq server. The `modelType` property was provided when the model was created.

The model name can be any string, so long as the first character is not a `/`.

Note that the resource instance URI and the `/tfmodels` URI always require authentication, even for public models.

Access Rights

- *admin* – admin users have full access rights to all TensorFlow model instances in their namespace.
- *user* – users have full access rights to any TensorFlow model instance which they have created.

Examples

Edge: Create or Update TensorFlow model

```
POST https://dev.vantiq.com/api/v1/resources/tensorflowmodels?modelName=sampleModel&modelType=tensorflow/yolo
<your boundary marker>
Content-Disposition: form-data; name="sampleModel"; filename="mymodel.bin"
Content-Type: application/octet
<bytes for the model>
<your boundary marker>
Content-Disposition: form-data; name="sampleModel"; filename="mymeta.json"
Content-Type: application/json
<bytes for the meta information>
<your boundary marker>
```

The headers:

```
ContentLength: Varies, based on boundary & model bytes length
ContentType: multipart/form-data; boundary=dLV9Wyq26L_-JQxk6ferf-RT153LhOO
```

Create TensorFlow model

```
POST https://dev.vantiq.com/api/v1/resources/system.tensorflowmodels
{
  "name": "myModel",
  "modelType": "tensorflow/plain",
  "contentSize": 203959508
}
```

(Note that this is not an atypical content size.)

This would be followed by the loading of the model content using tus.

Delete model

```
DELETE https://dev.vantiq.com/api/v1/resources/system.tensorflowmodels/sampleModel
```

Tests

Tests are the mechanism for testing the functionality of Vantiq resources or for a full Vantiq application.

The resource model for tests can be found in the Test Reference Guide.

Custom Operations

Relationships

Tests may have any of the following relationships:

- *test_for* relationship to the Rule, Procedure, or Project being tested
- *dependency* relationship to the setup and cleanup Procedures as well as any resources used as Test inputs, outputs, or as validation Procedures

Access Rights

- *admin* – admin users have full access rights to all test instances in their namespace.
- *user* – users have no access rights to test instances.

Test Reports

Test Reports are the test history for previously run tests. They contain the following properties:

- **id** – the unique identifier for the report
- **name** – the name of the test that was run
- **status** – the current status of the test which may be: *preparing, generating, executing, analyzing, complete, or cleaned*.
- **successes** - the list of expected outputs that were successfully received by the test
- **failures** – the list of expected outputs that were not received by the test or errors that occurred in the namespace during the test
- **startTime** – the time the test began to execute
- **endTime** – the time that test execution completed
- **isUnitTest** – a boolean that is true if the tested resource was a Rule or Procedure
- **testSuiteName** – if the test run was part of a larger test suite, the name of the test suite that was run

Test execution is triggered by performing an insert operation on the system resource `system.testreports`. To begin a Test run of a Unit or Integration Test, INSERT into the Type `system.testreports` with the `name` property set to the name of the Test. For example, to run a Test called `MyProcedureTest`:

```
Method: POST
URL: /api/v1/resources/system.testreports
Body: { "name": "MyProcedureTest"}
```

For more information, reference the Test Reference Guide

Access Rights

- *admin* – admin users have full access rights to all test report instances in their namespace.
- *user* – users have no access rights to test report instances.

Test Suites

Test Suites are the mechanism for running a series of tests that all test the same Rule, Procedure, or Project.

The resource model for Test Suites can be found in the Test Reference Guide.

Custom Operations

Relationships

Test Suites may have any of the following relationships:

- *test_for* relationship to the Rule, Procedure, or Project being tested
- *test_for* relationship to all of the tests contained by the suite

Access Rights

- *admin* – admin users have full access rights to all test suite instances in their namespace.
- *user* – users have no access rights to test suite instances.

Tokens

Tokens are one of the mechanisms for authenticating requests to the Vantiq Server. Tokens can be used to create long lived secure connections between nodes in a distributed Vantiq system. In order to connect two Vantiq nodes so that one node can issue requests to the other you can use either an access token or username/ password pair.

Tokens have the following properties:

- **accessToken** – an immutable, system generated value which is used as a natural key. Only visible to the creator of the token.
- **profiles** – a list of profile names which specify the authorizations associated with this token. Any profiles assigned cannot exceed the authorizations of the requester. If not specified, then the token will be a “personal token”.
- **singleNamespace** – a boolean indicating whether the token is limited to accessing the current namespace. Only relevant for personal tokens, as other tokens never have permissions outside their original namespace.
- **name** – a human-readable name that can be used to identify a token
- **username** – the user who this token was created for. If this is a token for connecting nodes, not a specific user, it will have a username like NODENAME__NAMESPACENAME.
- **expiresAt** – a DateTime value which indicates when the token “expires” (meaning that it can no longer be used as a valid authentication credential). By default tokens do not expire.
- **tokenType** – Optional, can either be ‘temporary’ or ‘permanent’. Users of the API need not specify this, we will assume by default tokens created through the API will be used by nodes, and will have a tokenType of ‘permanent’.

Note that when creating a token, the accessToken and expiresAt properties cannot be explicitly set. The accessToken property will automatically be generated by the system when the token is first inserted. The expiresAt property can be configured through use of the pseudo-property tokenTimeout, which can only be specified on insert. Use tokenTimeout to express how long the token should be valid, in seconds, and the token created will have an expiresAt time that is that far in the future.

A “personal token” is a token created with no profiles specified. Any user of a personal token will act as if they were the token creator, including access to all their namespaces. When creating a personal token, the **singleNamespace** property can be specified to limit the token to only the current namespace.

Access Rights

- *admin* – admin users have full access rights to all Token instances in their namespace. They cannot see the **accessToken** field for tokens created by other users.
- *user* – users have read-only access rights to all Token instances which they have created (although users cannot directly create their own tokens, there are situations where the system will create them on their behalf).

Examples

Create Access Token

This creates a new token with the “user” profile (from the requestors namespace) which will expire 10 minutes (600 seconds) after creation.

```
Method: POST
URL: /api/v1/resources/tokens
Body: { "profiles": ["system.user"], "name": "testToken", tokenTimeout: 600 }
```

Delete token or revoke access

This deletes (revokes) a previously created token

```
Method: DELETE
URL: /api/v1/resources/la28DUJ008LLAJllsliqo
```

(where la28DUJ008LLAJllsliqo is a valid token string)

Topics

User-defined topics for use in the Vantiq publish/subscribe model may be registered, published and deleted with the Vantiq services. Registration is optional but recommended as it enriches the metadata on which the Vantiq IDE operates.

Topics have the following properties:

- **name** – the name of the topic
- **description** – the topic's description
- **isReliable** – indicates whether the events received by the topic will be handled "reliably".
- **autoUnwind** – indicates whether the system should automatically "unwind" any event data received by the topic. If `true`, then if an array is received as event data, the system will immediately process the array and generate one event for each array member (instead of one containing the array itself).

Topic names are always prefixed with a leading `/` and contain only alphanumeric characters, with optional `/` separators. Each `/` character must be followed by at least one alphanumeric character, the topic string may not end in a `/`. Topic names may not start with: `/type`, `/property` or `/system`.

> When specifying a topic as a resource name on the URL, the final URL always contains '//' at the start of the topic name. The first '/' identifies the next part of the URL and the second '/' the first character of the topic name.

Custom Operations

Topics support the following custom resource operations.

Publish

A topic is published by issuing a publish request on the topic name with the body of the request containing the data associated with the publication. This data value will be delivered to any subscribed rule as the **newValue** property of the event.

```
POST https://dev.vantiq.com/api/v1/resources/topics/<topicName>
{
    <eventData>
}
```

Relationships

Topics may have a *dependency* relationship to the Type used as the messageType.

Access Rights

- *admin* – admin users have full access rights to all Topic instances in their namespace.
- *user* – users have read-only access rights to all Topic instances in their namespace.

Examples

Create Topic

```
POST https://dev.vantiq.com/api/v1/resources/topics
{
  "name": "/my/sample/account/update/topic",
  "description": "Published after a customer successfully updates their account info."
}
```

Publish Topic

```
POST https://dev.vantiq.com/api/v1/resources/topics//weather/denver/hourly
{
    "temp": 49,
    "wind": 2,
    "barometer": 990
}
```

Delete Topic

```
DELETE https://dev.vantiq.com/api/v1/resources/topics//weather/denver/hourly
```

Note that the topic name must be URL encoded in order to be placed in the DELETE URL.

TrackingRegions

A tracking region represents a named area in a coordinate space. Tracking regions have the following properties:

- **name** – this is the name of the tracking region.
- **boundary** – this is an Object that contains a list of (at most 4) points that comprise the boundary of the region.
- **distance** – this is an Object containing the following properties:
  - **points** – a list of two points
  - **distance** – the distance between these two points
- **direction** – this is an Object containing the following properties
  - **points** – a list of two points
  - **direction** – the direction (compass degrees between 0 and 360) that movement from the first point to the second represents.

In each of the above, points are specified with an **x** and **y** component (alternately, you can use **lon** and **lat**, respectively.

A special note on the boundary: the boundary forms a polygon from which we determine if an object is *inside*. Because of this, the order of the points is important. If you consider walking counter-clockwise from the first point to the second, and so on, the *inside* of the boundary is considered to be *on your left*. That is, you will specify the boundary in a counter-clockwise fashion. If the boundary points are not in a reasonable order, you will get an error stating that the polygon formed has either zero or infinite area. To resolve this, correct the order of the boundary points.

The `distance` and `direction` properties are optional. If they are not present, motion tracking cannot determine velocity information.

The JSON representation of a tracking region as follows:

```
{
    "name": "intersection",
    "boundary":
        [
            { "x": 670, "y":699 },
            { "x": 1767, "y": 700 },
            { "x": 1765, "y": 1066 },
            { "x": 671, "y": 1067 }
        ],
    "distance": {
        "points": [
            { "x": 746, "y": 418 },
            { "x":1033, "y": 420 }
        ],
        "distance": 24
    },
    "direction": {
        "points" : [
            { "x": 523, "y": 1000 },
            { "x": 623, "y": 1000 }
        ],
        "direction": 90
    }
}
```

In this example, we have a region called *intersection* with a defined boundary. The distance between the two points specified is 24 (the units are unspecified, but should be understood by the application developer). The direction between the two points listed is east (compass direction 90 degrees).

For more information about using regions, please see the Motion Tracking section of the Image Processing Guide.

Access Rights

- *admin* – admin users have full access rights to all Video instances in their namespace.
- *user* – users have full access rights to any Video instance which they have created.

Examples

Create Tracking Region

```
POST https://dev.vantiq.com/api/v1/resources/system.trackingregions
{
    "name": "intersection",
    "boundary":
        [
            { "x": 670, "y":699 },
            { "x": 1767, "y": 700 },
            { "x": 1765, "y": 1066 },
            { "x": 671, "y": 1067 }
        ],
    "distance": {
        "points": [
            { "x": 746, "y": 418 },
            { "x":1033, "y": 420 }
        ],
        "distance": 24
    },
    "direction": {
        "points" : [
            { "x": 523, "y": 1000 },
            { "x": 623, "y": 1000 }
        ],
        "direction": 90
    }
}
```

Delete Tracking Region

```
DELETE https://dev.vantiq.com/api/v1/resources/system.trackingregions/intersection
```

Types

Types represent the data (objects) managed by the Vantiq System. Types are classified as either **resource** types or **schema** types.

- Resource types, also known simply as **types** contain collections of identically typed object instances.
- Schema types are used to describe the structure (schema) of data obtained from an external source or transmitted in an event.

A type has the following properties:

- **name** – the name of the type as a String. The name must be unique. Type names starting with “Ars” are reserved for system use to unambiguously identify system types such as ArsChat and ArsRuleSnapshot. A type name is an alphanumeric string which may contain embedded underscore characters. No other characters are allowed in a type name. A type name may be prefixed with a namespace name.
- **role** – one of:
  - **standard** – the type defines a user-defined resource. If not specified, **standard** is the default.
  - **schema** – the type defines a schema type.
- **version** – the version of the type definition as a String. This property is reserved for future use.
- **documentation** – an array of links to documentation resources describing the type.
- **properties** – a map describing the properties defined on the type. Each entry in properties is an object containing the following properties:
  - **name** – property names starting with “_” or “ars_” are reserved for system use. Property name is an alphanumeric string which may contain embedded underscore characters.
  - **type** – One of the following built-in types: [“String”, “Integer”, “Real”, “Boolean”, “DateTime”, “Decimal”, “Currency”, “GeoJSON”, “Object”] or name of a user-defined type or schema.
  - **required** – a boolean indicating whether the value must be supplied on create or insert.
  - **default** – the default value to assign if the property is not referenced in the insert request. A default value of **null** is equivalent to not specifying the default property at all as it is not possible to declare a default value that is the null value.
  - **scale** – for Decimal and Currency values the number of digits of precision being carried.
  - **array** – boolean indicating that the property is multi-valued and, therefore, represented as an array of values.
  - **encrypted** – boolean indicating the value of the property will be encrypted when stored in the database. The encryption key defined for the namespace will be used to encrypt the value. Note: the value of **encrypted** should **NOT** be changed if there are instances of the type stored in the database - once set, an attempt will be made to decrypt all values of this property leading to corrupted data being returned for those instances which pre-dated the enabling of encryption on this property.
- **maxStorage** – The maximum amount of storage (in bytes) to be allocated to this type. If more objects are inserted into the type than will fit in the allocated space, old objects are removed using a first in, first out strategy to make space available for the newly inserted objects. The space allocated must be at least 4096 bytes. Note that there are two restrictions on the operations allowed on objects in a type with maxStorage value:
  - updates are not supported
  - delete is not supported
- **maxObjects** – The maximum number of object instances to be stored for the type. This property may only be set if maxStorage is set. If maxObjects is reached before maxStorage has been reached, objects will be removed from the type using a first in, first out strategy to make room for the new objects. The maxStorage value always takes precedence over maxObjects. Therefore, there will only maxObjects in the type if there is sufficient storage allocated by maxStorage AND at least maxObjects have been inserted into the type.
- **naturalKey** – an array of property names identifying the properties that make up a unique key for the objects stored in the type. The naturalKey may contain the names of one or more properties. It is the responsibility of the user to guarantee that objects with identical naturalKeys are not inserted into the type. One option for enforcing this constraint is to define a unique index on the primary key properties. Another option is to use

only the UPSERT statement to add new objects to type. The UPSERT statement will always peform an update to the existing object if it finds an object with the primary key value specified in the UPSERT.

- **indexes** – an array of indexes to be defined on the type. Each entry is an index description consisting of:
  - **keys** – an array of property names defining the (composite) key. The name is preceded by a '-' if the key should be indexed descending and an optional '+' if the key should be indexed ascending. The name is preceded by a '#' if the key is a geolocation (represented in GeoJSON format) and a geospatial index is requested. Only a single property in the key set can be declared as a geospatial key. The values of the geospatial key will be interpreted as points, lines and areas on the surface of a sphere.
  - **options** – an object containing any options to be applied to the index. Current options are:
    - **unique** – a boolean value indicating whether the index is unique.
- **foreignKeys** – identifies properties whose values represent the natural key of another type. Such keys supports references from one type to another type. The foreignKey property contains an array of foreign key definitions. Each foreign key definition contains the name of the target type and an array of key properties with each key property consisting of a pair representing the property name in the local type and the corresponding property name in the target type. Example:

  { "name": "MyExampleEmp", "properties": { "empId": { "type": "String" }, "empAddress": { "type": "String" }, "deptName": { "type": "String" } }, "foreignKeys": [{"type": "dept", "keys": [{"property":"deptName", "targetProperty": "name"}]] }

Assuming **dept** has the following minimal definition:

```
{ "name: "dept",
  "properties": {
     "name": { "type": "String" }
   },
  "naturalKey": ["name"]
}
```

- **userAccessLevel** – specifies the level of access that should be granted to the type by the built-in `user` profile. The possible values are the well-known system access levels or one of the type-specific custom levels.
- **adminAccessLevel** – specifies the level of access that should be granted to the type by the built-in `admin` profile. The possible values are the well-known system access levels or one of the type-specific custom levels.
- **accessLevels** – specifies additional access levels that can be used when determining what permissions should be granted for a type. The format is a mapping object where the key is the name of the access level and the value is a profile DML entry. Example:

```
{
  "myAccessLevel": {
      "select": {},
      "insert": {},
      "update": {"total" : {"ars_substitution_lt": "USD:100"}},
      "delete": {},
      "aggregate":{},
      "invoke": []
  }
}
```

- **expiresAfter** – specifies a time to live for instances of this type as an interval constant (e.g. `1 hour` or `2 days`). The time is measured from the moment that the instance is created (as recorded in the `ars_createdAt` system property) and once the instances has "expired" the system may delete the instance at its discretion. Instances that remain beyond their expiration time will function identically to those that have not (so if your application needs to ignore them, it should remove them explicitly).

- **rulesSuppressed** – when `true` specifies that the runtime will not execute rules which trigger on insert, update, or delete events for this type. This applies to both `BEFORE` and `WHEN` rules. For types in applications that do not require rule-based processing to occur on write operations, the system can greatly reduce the overhead and realize a significant performance improvement when executing them.
- **storageManager** – a reference to the storage manager repsonsible for handling all data requests related to this type. See storagemanagers. The reference consists of the following fields:
  - **name** – the name of the storage manager
  - **connection** (optional) – the name of the connection the storage manager must use for this type. If no connection is provided, then the default connection is used.
  - **properties** (optional) – an Object containing name-value pairs defining any additional storage manager specific properties. These are provided as-is to the storage manager when manipulating the type.

JSON Representation

A stylized JSON representation of a type object:

```
{ "name": "String",
  "properties": {"<propertyName>": {"type": "String", "required": "Boolean"},
           "<propertyName>": {"type": "String", "required": "Boolean"},
           ...
           },
  "indexes": [{"keys": [<"propertyName>", "<propertyName>", ...], "options": {"unique": "Boolean"}}]
}
```

Deleting Types

When a type is deleted, all instances of the type are deleted.

Compatible Type Extensions

Type definitions may be modified by authorized users. However, care must be taken if instances of the type exist at the time the type definition is modified. Vantiq recommends that such type changes be upward compatible with existing instances of the type:

- add a property
- modify index definitions

Incompatible extensions include changing the attributes of a property. For example, if the type for a property is changed from **Integer** to **String**, the data in any existing instances of the type will not be converted to string and will generate unexpected errors when referenced. Similarly, if the **multi** attribute for a property is changed, the internal representation of existing instances of the type will be incorrect and will generate runtime errors.

The **name** and **role** attributes of a type may not be changed. The type must be re-created in order to change one of those attributes.

Query Substitutions

Types support queries to identify the set of objects returned by a GET request. In order to make queries reasonably general, certain values known to the system at runtime must be available for use in queries. Such parameters may be included in a query constraint by specifying the parameter in the following form:

```
{"ars_substitution": <value>}
```

where `<value>` is a special reserved identifier.
Special reserved identifiers are available for these system-known values:

- `ars_username` – refers to the name of the currently authenticated user. Helpful when creating audit objects.
- `ars_currentDateTime` – refers to the current date and time (UT). Helpful when timestamping objects.
- `ars_substitution_<operator>` – this is a highly specialized pattern that defines a query constraint operator to be substituted into the query. `<operator>` can be replaced by any query operator such as ‘or’, ‘and’, ‘elemMatch’, etc. That is, by simply naming the operator but without the leading ‘$’. The operator is then substituted into the query as `$<operator>`. The reason for this construct is that the default storage manager (MongoDB) will not store property names containing a leading ‘$’. By replacing the leading $ with ‘ars_substitution_’, the constraint can be stored in the database. Note that it is the user’s responsibility to specify the query constraint in a valid format.

Example:

```
{"name":{"ars_substitution": "ars_username" }}
```

Custom Operations

Generating JSON Schema

To produce a JSON Schema representation of a type, use the existing `selectOne` operation and set the `contentType` option to `application/json+schema`. You can also use the URI `resources/types/<typeName>/_schema` to produce the schema form of a specific type instance.

Relationships

Types have a *dependency* relationship to any nested Types referenced within the Type definition.

Rename

Types support the *rename* operation which will change the name of a type without deleting its contents. To maintain the constraint that names are immutable, this is actually treated as a delete operation, followed by an insert operation (during which the expected type events are generated). The system ensures that the data associated with the type is transferred in the most efficient manner possible.

Access Rights

- *admin* – admin users have full access rights to all Type instances in their namespace and read-only access to any system types.
- *user* – users have read-only access rights to all Type instances.

Group Enabled

Types are a “group enabled” resource. Placing a “type” into a group restricts use of the type to identities which are members of the group.

Events

Events are generated for the following operations:

- **insert** – occurs any time an instance of a type is inserted. The data provided is the instance of the type that was inserted.

- **update** – occurs any time one or more instances of a type are updated. The data provided depends on whether this was a single instance update or a bulk update:
  - For a single instance update the data is the updated instance.
  - For a bulk update the data is an object with the following properties:
    - ars_bulkOperation – boolean with a value of `true`.
    - qual – an object describing the query used to select the instances to udpate.
    - count – the number of instances updated.
- **delete** – occurs any time one or more instances of a type are deleted. The data provided depends on whether this was a single instance delete or a bulk delete:
  - For single instance delete the data provided is the deleted instance.
  - For a bulk delete the data is an object with the following properties:
    - ars_bulkOperation – boolean with a value of `true`.
    - qual – an object describing the query used to select the instances to udpate.
    - count – the number of instances updated.

Examples

Create Type

```
POST https://dev.vantiq.com/api/v1/resources/types
{
  "name": "PaulType",
  "properties": {
      "empName": { "type": "String" },
      "salary": {"type": "Integer" },
      }
  "indexes": [{keys: ["empName"]]
}
```

Delete Type

```
DELETE https://dev.vantiq.com/api/v1/resources/types/PaulType
```

Users

A User resource defines users authorized to access Vantiq services. Each namespace supports a set of users. Users are provisioned by a namespace administrator using the POST, PUT, PATCH and DELETE REST requests.

A User resource has the following JSON representation:

```
{
  "username": "<myUserName>",
  "password": "<myClearTextPassword>",
  "profiles": ["<profile1>", ...],
  "email": "<myEmailAddress>",
  "preferredUsername": "<preferredUsername>",
  "firstName": "<myFirstName>",
  "lastName": "<myLastName>"
}
```

The `username` property is required and must be globally unique for the Vantiq deployment. User names may contain the following characters:

- alphanumerics
- embedded underscore(_)
- embedded period(.)
- embedded dash(-)
- embedded atSign(@)

This covers the most common characters used in email addresses. By convention, user names are generally defined as one of the user's email addresses.

User names are case insensitive. Note that all other names in Vantiq are case sensitive.

The `password` is also required and once transmitted during the initial user creation is stored as a one-way hash value.

When configured to use an OAuth based authorization provider, both the `email` and `preferredUsername` properties will be automatically populated using the OAuth profile claims (when available).

The `profiles` property is optional, but if no value is provided the system will assign the built-in `user` profile for both local and system access.

All other properties are optional.

Access Rights

- *admin* – admin users have full access rights to all User instances in their namespace. This includes the ability to explicitly set passwords on non-OAuth systems.
- *userAdmin* – userAdmin users have full access rights to all User instances in their namespace. This includes the ability to explicitly set passwords on non-OAuth systems.
- *developer* – developers have read and update access rights to their own User instance. They are also restricted from changing their own profiles.
- *user* – users have read and update access rights to their own User instance. They are also restricted from changing their own profiles.

Events

Events are generated for the following operations:

- **authenticationSuccess** – occurs whenever the user is successfully authenticated. Only applies when Vantiq is configured to use its internal authentication system. The event is always generated in the user's home namespace. The data provided on the event is the users instance.
- **authorizationFailure** – occurs whenever the user attempts an operation for which they are not authorized. The data provided is an object with the properties:
  - error – the error message
  - namespace – the namespace in which the failure occurred.

Examples

Create User

```
POST https://dev.vantiq.com/api/v1/resources/users
{
  "username": "paul@paulsdomain.com",
  "password": "mypassword"
}
```

Creates a new user named "paul@paulsdomain.com" with the password specified. In this simple example all other properties are assigned their default values.

Delete User

```
DELETE https://dev.vantiq.com/api/v1/resources/users/paul@paulsdomain.com
```

Videos

A video represents the data comprising a video stored in the Vantiq Database. Video instances are created by uploading the contents of the video using a multi-part MIME message of the following form:

```
POST https://dev.vantiq.com/api/v1/resources/videos
<your boundary marker>
Content-Disposition: form-data; name="<assignedName>"; filename="<filename>"
Content-Type: <contentType>
<file content>
<your boundary marker>
```

where:

- `<assignedName>` is the name of the video as stored in **videos** instance.
- `<fileName>` is the name of the file from which the content was obtained.
- `<contentType>` is the mime type of the video (*e.g.* `video/mpeg`, `video/mp4`)

In addition, the appropriate HTTP headers must be set:

```
ContentLength: <length of http content in above message>
ContentType: multipart/form-data; boundary=<your boundary marker>
```

Once the file has been uploaded its meta-data will be available via the resource URI: `api/v1/resources/videos/<assignedName>`.
This will return the following JSON representation:

```
{
    "name": "assets/movies/cars/mydrive.mp4",
    "fileType": "video/mp4",
    "content": "/vids/assets/movies/cars/mydrive.mp4"
}
```

The `content` property is the absolute URI that can be used to retrieve the file's content from the Vantiq server. The `fileType` property is set by Vantiq based on the uploaded file's extension, but it can be changed once the file has been uploaded.

The video name can be any string, so long as the first character is not a `/`.

Note that the resource instance URI and the `/vids` URI always require authentication, even for public videos.

Create Video from Temp Blob

If you have obtained a `TempBlob` (*e.g.*, from a REMOTE source), you can create a video from that information. To do this, use the `fromTemp` property, specifying the Resource Reference of the Temp Blob.

```
Method: POST
URL: /api/v1/resources/videos
Body: { "name": "<copyName>",
        "fromTemp": "<Temp Blob reference>"
      }
```

Access Rights

- *admin* – admin users have full access rights to all Video instances in their namespace.
- *user* – users have full access rights to any Video instance which they have created.

Examples

Create or Update Video

```
POST https://dev.vantiq.com/api/v1/resources/videos
<your boundary marker>
Content-Disposition: form-data; name="sampleVideo"; filename="mydrive.mp4"
Content-Type: video/mp4
<bytes for the video>
<your boundary marker>
```

The headers:

```
ContentLength: Varies, based on boundary & video bytes length
ContentType: multipart/form-data; boundary=dLV9Wyq26L_-JQxk6ferf-RT153LhOO
```

Delete Video

```
DELETE https://dev.vantiq.com/api/v1/resources/videos/sampleVideo
```

References

Kubernetes is a registered trademark of The Linux Foundation.