Shovel plugin

The *Shovel plugin* allows you to configure a number of *shovels*, which start automatically when the broker starts.

The high level goal of a *shovel* is to reliably and continually take messages from a queue (a *source*) in one broker and publish them to exchanges in another broker (a *destination*).

The source queue and destination exchanges can be on the same broker or distinct brokers.

A shovel behaves like a well-written client application, which connects to its source and destination, reads and writes messages, and copes with connection failures.

The primary advantages of a shovel are:

Loose coupling
: A shovel can move messages between brokers (or clusters) in different administrative domains:

- they may have different users and virtual hosts;
- they may run on different versions of RabbitMQ and Erlang.

WAN-friendly
: The Shovel plugin uses AMQP to communicate between brokers, and is designed to tolerate intermittent connectivity without message loss.

Highly tailorable
: When a shovel connects (either to the source or the destination) it can be configured to perform any number of explicit methods. For example, the source queue need not exist initially, and can be *declared* on connect.

A comparison between *clustering*, *federated exchanges* and *shovels* is given on the **Distributed Messaging** page.

What does it do?

The Shovel plugin defines (and runs) an Erlang client application for each shovel.

In essence, a shovel is a simple pump. Each shovel:

- *connects* to the source broker and the destination broker,
- *consumes* messages from the queue,
- *re-publishes* each message to the destination broker (using, by default, the original exchange name and routing_key).

The shovel configuration allows each of these processes to be tailored.

connects
: After connection to a source or a destination broker a series of configured AMQP *declarations* can be issued. Queues, exchanges and bindings can be declared.

A shovel will attempt to reconnect to a broker if a failure occurs and multiple brokers can be specified for the source and destination so that another broker may be selected (at random) to reconnect to. A reconnection delay can be specified to avoid flooding the network with reconnection attempts, or to prevent reconnection on failure altogether.

All configured *declarations* (for that source or destination) are re-issued upon re-connect.

consumes
: The shovel's consumer can acknowledge messages automatically on receipt, after (re-)publication, or after confirmation of its publication.

re-publishes
: Both the publish method and the message properties can be modified with explicit parameter values.

Getting started

The *Shovel plugin* is included in the RabbitMQ distribution. To enable it, use **rabbitmq-plugins**:

```
rabbitmq-plugins enable rabbitmq_shovel
```

You may also wish to enable the `rabbitmq_shovel_management` plugin (see **below**).

There is no requirement to run the shovel on the same broker (or cluster) as its source or destination; the shovel can be entirely separate.

There are two distinct ways to define shovels: **static shovels** and **dynamic shovels**. The differences are as follows:

Static Shovels	Dynamic Shovels
Defined in the broker **configuration file**.	Defined in the broker's **parameters**.
Require a restart of the hosting broker to change.	Can be created and deleted at any time.
Slightly more general: any queues, exchanges or bindings can be declared manually at startup.	Slightly more opinionated: the queues, exchanges and bindings used by the shovel will be declared automatically.

Shovels between clusters

In This Section
- Server Documentation
- Client Documentation
- Plugins
  - Management plugin
  - Federation plugin
  - Shovel plugin
    - Static Shovels
    - Dynamic Shovels
  - STOMP plugin
  - MQTT plugin
  - LDAP plugin
  - Configuring web plugins
  - Community plugins
  - Installing plugins
  - Plugin development
- News
- Protocol
- Our Extensions
- Building
- Previous Releases
- License

In This Page
- What does it do?
- Getting started
- Shovels between clusters
- Monitoring shovels

Related Links
- AMQP URI Spec
- AMQP URI Params

It's normally desirable to ensure that shovels are resilient to failure of any node in the source or destination clusters, or the cluster hosting the shovel.

You can ensure that shovels can fail over to different nodes in the source or destination clusters by specifying multiple source and/or destination URIs for each shovel.

Dynamic shovels are automatically defined on all nodes of the hosting cluster on which the shovel plugin is enabled. Each shovel will only start on one node (arbitrarily), but will fail over to another node if that one goes down.

Static shovels should be defined in the configuration file for all nodes of the hosting cluster on which the shovel plugin is enabled. Again each shovel will only start on one node and fail over when needed.

Monitoring shovels

There are two ways of discovering the status of shovels.

Use shovel management

Shovel status can be reported on the **Management plugin** user interface by enabling the `rabbitmq_shovel_management` plugin wherever you have the management plugin enabled.

Information about configured shovels will automatically appear in the management API and UI.

Direct query

Shovel status can be obtained by direct query of the Shovel plugin app. Issue the following `rabbitmqctl` command:

```
$ rabbitmqctl eval 'rabbit_shovel_status:status().'
```

This calls the `status` method in a module of the `rabbitmq_shovel` plugin, which will return an Erlang list, with one element for each configured shovel.

Each element of the list is a tuple with four fields: `{`*`Name`*`, `*`Type`*`, `*`Status`*`, `*`Timestamp`*`}`.

- *`Name`* is the shovel name,
- *`Type`* is either `static` or `dynamic`,
- *`Status`* is the current shovel state,
- and *`Timestamp`* is the time when the shovel *entered* this state.

`Timestamp` is a local calendar time of the form `{{YYYY, MM, DD}, {HH, MM, SS}}`.

`Status` takes one of three forms:

- The shovel is starting up, connecting and creating resources:

  ```
  'starting'
  ```

- The shovel is running normally:

  ```
  { 'running', {'source', Source},
               {'destination', Destination}}:
  ```

  where *`Source`* and *`Destination`* terms give the respective connection parameters.

- The shovel has terminated:

  ```
  {'terminated', Reason}
  ```

  where *`Reason`* is an Erlang term that indicates the reason for the termination.