Feature 1266 Rule Graph API

- Status: Design
- Author: Jiyong Huang
- Discussion: https://github.com/lf-edge/ekuiper/issues/1266

Motivation

Currently, SQL is the only way to describe the business logic of a rule. There are some drawbacks:

- For non-technical person, SQL is hard to write and understand.
- For some scenarios, SQL is not expressive enough. For example, filter before or after a window; calculate but not select: `SELECT a * 200 AS anotherA, b, c FROM demo WHERE anotherA > b` ; and create complex pipeline.
- Not easy to generate from UI.

To complement the SQL representation, we plan to provide another business logic presentation model: a graph model to describe the nodes and the DAG by JSON format. This is model which can directly map to a graph in the UI.

Notice that, the Graph API is parallel with SQL. The user can use both ways to create a rule. Internally, SQL queries are converted into a DAG. Thus, it is a subset of graph API.

Proposed API

We extend the rule API to add a field `graph`. It is mutual exclusive with `sql` field.

```
POST http://{{host}}/rules
Content-Type: application/json

{
    "id": "rulePv",
    "graph": {the json to describe a DAG}, || "sql":"",
    "options":{}  // optional
}
```

An example to create a rule with graph API.

```
POST http://{{host}}/rules
Content-Type: application/json

{
  "id": "rulePv",
  "name": "ui",
  "graph": {
    "nodes": {
      "abc": {
        "type": "source",
        "nodeType": "mqtt",
        "props": {
          "datasource": "demo"
        }
      },
      "myfilter": {
        "type": "operator",
        "nodeType": "filter",
        "props": {
          "expr": "temperature > 20"
        }
      },
      "logfunc": {
        "type": "operator",
        "nodeType": "function",
        "props": {
          "expr": "log(temperature) as log_temperature"
        }
      },
      "sinfunc": {
        "type": "operator",
        "nodeType": "function",
        "props": {
          "expr": "sin(temperature) as sin_temperature"
        }
      },
      "pick": {
        "type": "operator",
        "nodeType": "pick",
        "props": {
          "fields": ["log_temperature", "humidity"]
        }
      },
      "mqttpv": {
        "type": "sink",
        "nodeType": "mqtt",
        "props": {
          "server": "tcp://syno.home:1883",
          "topic": "result",
          "sendSingle": true
        }
      },
      "mqtt2": {
        "type": "sink",
        "nodeType": "mqtt",
        "props": {
          "server": "tcp://syno.home:1883",
          "topic": "result2",
          "sendSingle": true
        }
      }
    },
    "topo": {
      "sources": ["abc"],
      "edges": {
        "abc": ["myfilter", "sinfunc"],
        "myfilter": ["logfunc"],
        "logfunc": ["pick"],
        "pick": ["mqttpv"],
        "sinfunc": ["mqtt2"]
      }
    }
  }
}
```

Graph model

The graph model is represented by JSON. It is composed by two part:

- nodes to define each node in the graph with properties.
- topo to define the pipeline.

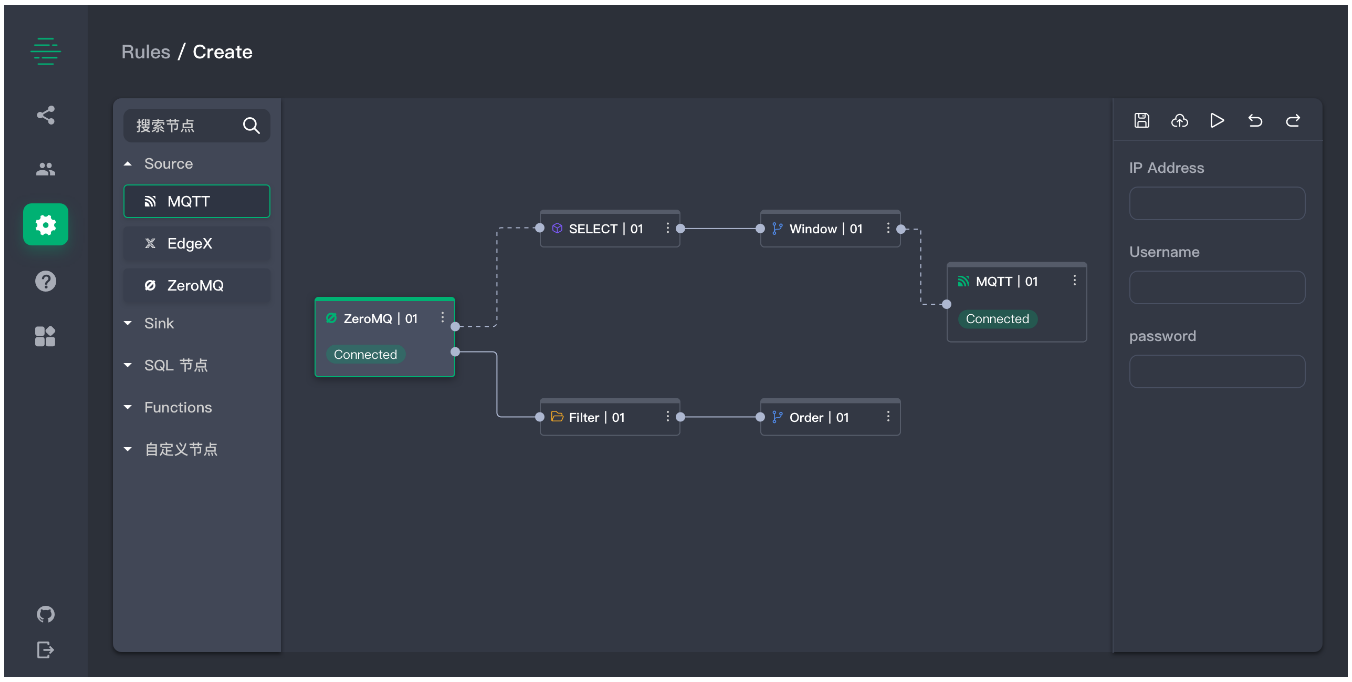
Nodes

All the source/sink and operator or functions can be presented as a node including built-in and plugin. Currently, we only provide nodes which is also shared by SQL. In the future, it is possible to provide more built-in nodes that cannot be mapped to SQL operators.

In the initial version, the below built-in nodes will be provided:

- Source
  - mqtt
  - edgex
  - zeroMq etc.
- Sink
  - mqtt
  - file
  - tdengine etc.
- Operator
  - Function
    - arithmetic
    - string etc.
  - Filter
  - Pick
  - Window
  - Join
  - Group

UI Draft


Rules / Create
搜索节点
Source
MQTT
EdgeX
ZeroMQ
Sink
SQL 节点
Functions
自定义节点
SELECT | 01
Window | 01
MQTT | 01
Connected
ZeroMQ | 01
Connected
Filter | 01
Order | 01
IP Address
Username
password