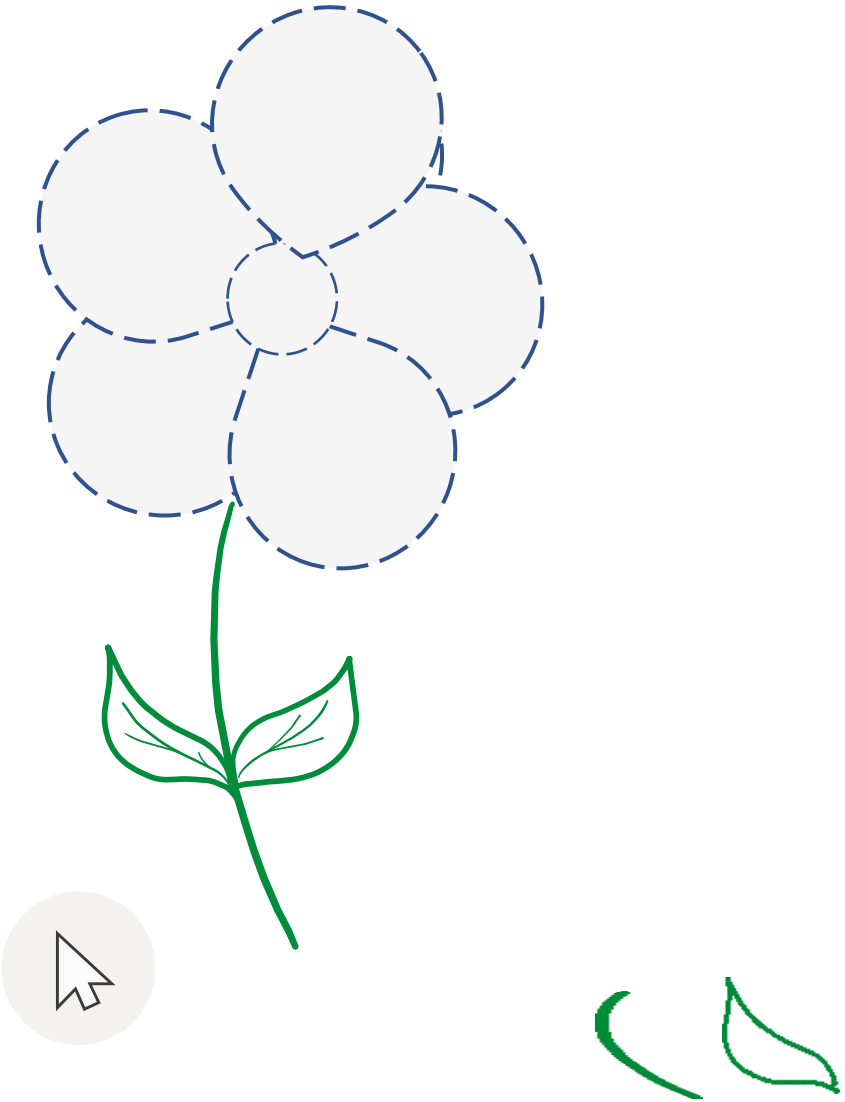
n8n.io

# Automation for SOCs with N8N

@ali_alwashali

# Agenda

1. The problem
2. Automation Five Principles
3. Security Automation Use Cases
4. How N8N Works
5. Automation Workflow Examples

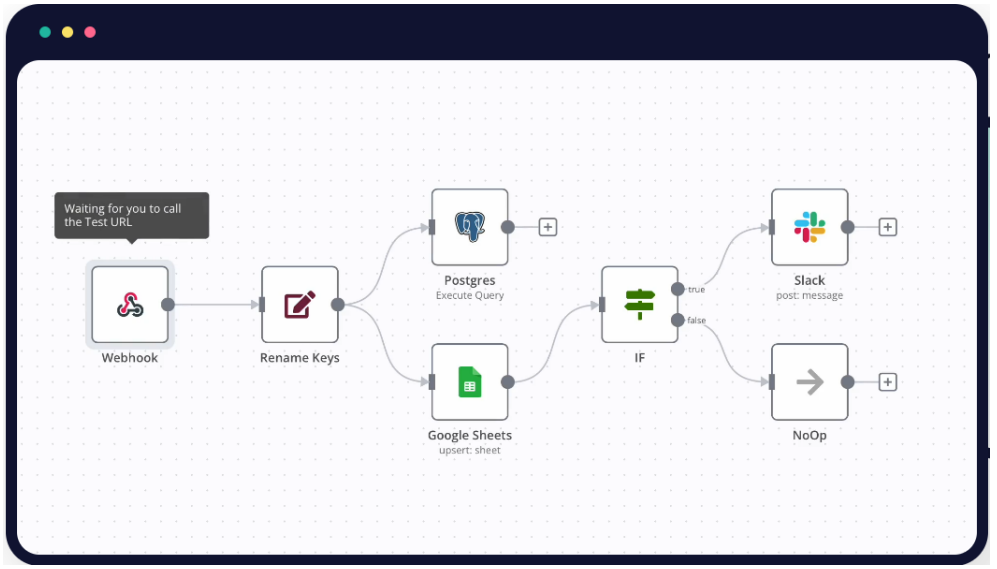


# The problem

- SOC spends 37% of its time on operational tasks & reporting
- 20% of alert investigation time is spent on triaging steps that can be fully automated
- Ticketing and Case Management
- Multi-Technology SOC Challenges
- Consistency: workflows are not humans, and they produce the same result each time.
- Workflows Faster than humans and don’t get bored
- You don’t have to have a problem to start automating

# Automation Five Principles

1. Automate all the things
2. Simplicity
3. Only orchestrate
4. Leverage Cloud Services
5. Minimize Coding



# Scripting vs Workflow Automation

## When to use an automation workflow instead of scripting ?

- Scripting is Automation ✓
- Productivity vs Customizations: How fast you want it ready!!
- Maintainability: When things get complex, can it be easily maintained by anyone in the team?
- Delegation: Are there more reliable services to handle the same task?

# SOAR/N8N

Automates processes ~~with no coding,~~ a little to almost no coding.

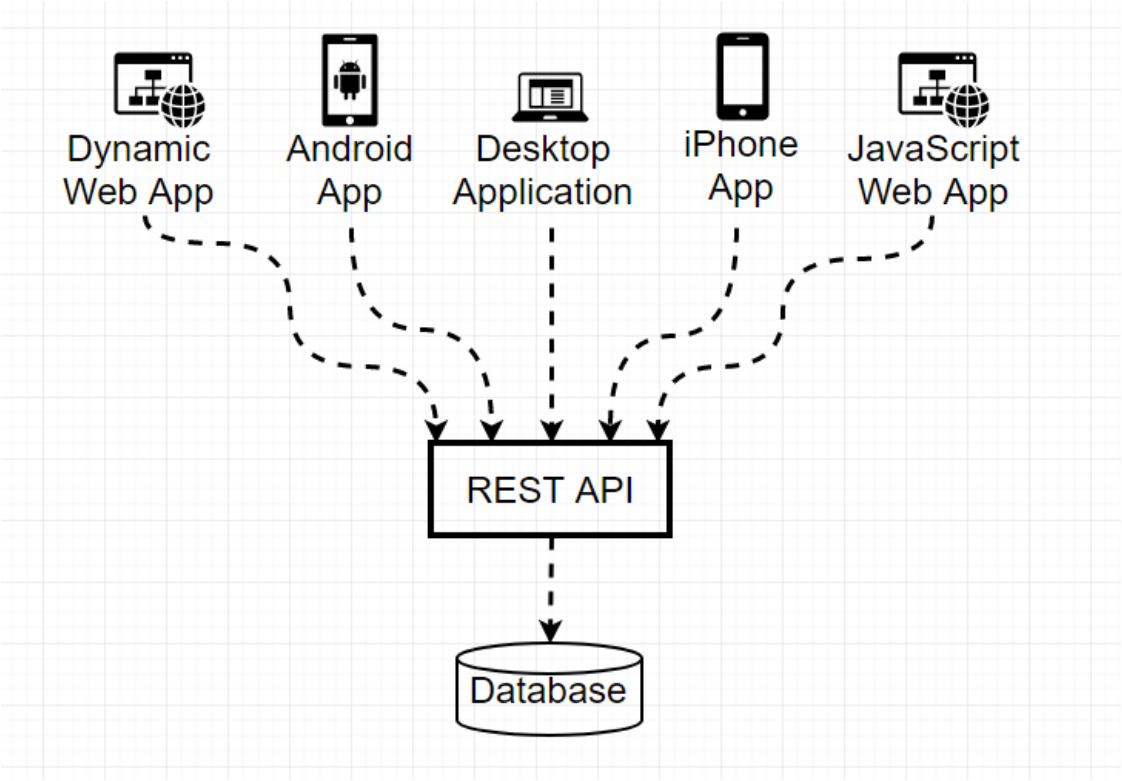
# Security Automation Use Cases

1. Intelligence Ingestion
2. SOC Reporting
3. Security Controls Integration
4. Automatic Alerts Closure
5. Intelligence/Data Enrichment
6. Security Operation
7. Automating existing investigation playbooks


**“ Any three wishes, except the automation of your legacy IT infrastructure.”**

# APIs

An **application programming interface** (**API**) is a way for two or more computer programs to communicate with each other.



# Everything is JSON

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format.

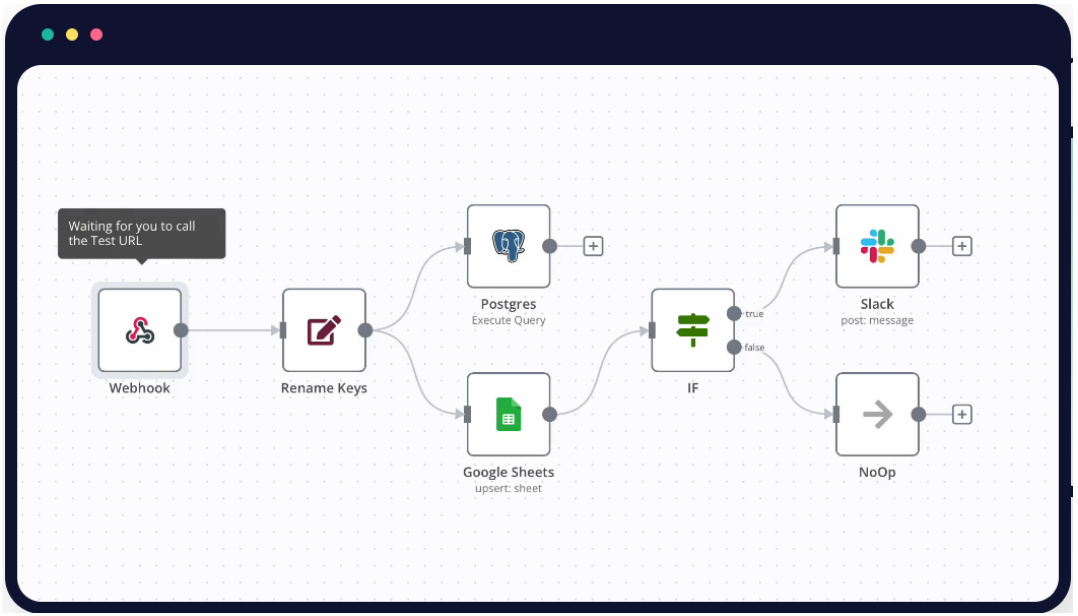
It is easy for humans to read and write. It is easy for machines to parse and generate.

# Everything is JSON

```
{
  "product": "N8N",
  "version": "0.1111",
  "releaseDate": "2022-06-21",
  "demo": true,
  "person": {
    "id": 12345,
    "name": "Jack",
    "phones": {
      "mobile": "877-123-1234"
    },
    "email": [
      "J@N8N.com",
      "J@yahoo.org"
    ],
    "dateOfBirth": "1980-01-02T00:00:00.000Z",
    "registered": true
  }
}
```

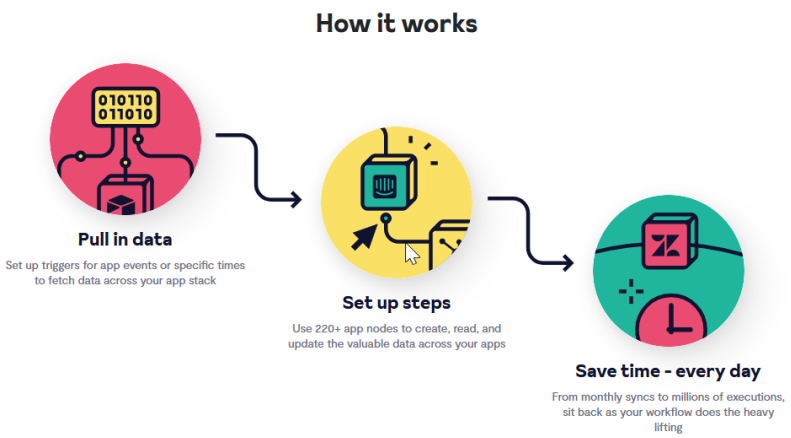
# n8n

- Easy to use
- 400+ Integrations https://n8n.io/integrations/
- Available for free



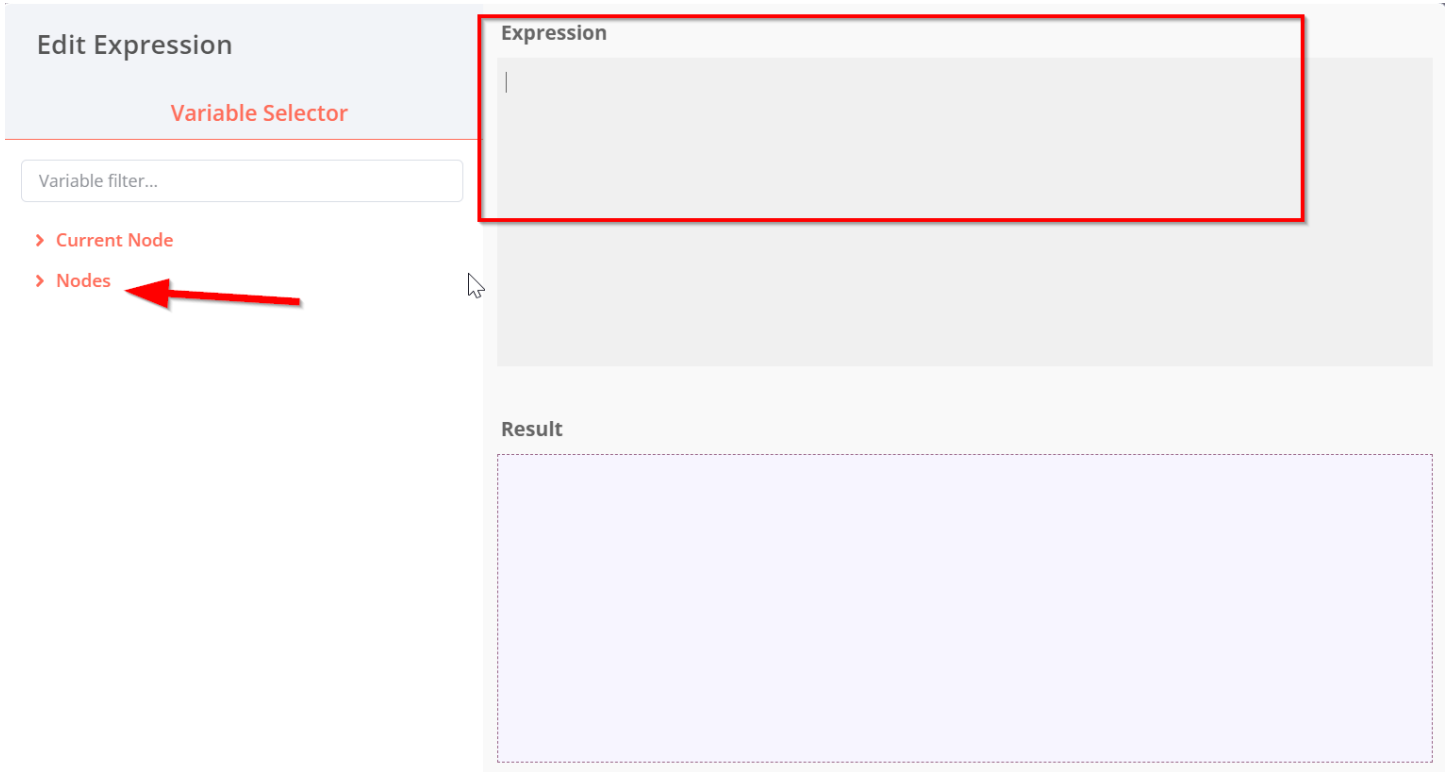
# How N8N works !!

- Workflow consists of nodes
- Each node takes input and produce output
- Nodes language is JSON
- Node output can serve many nodes, but a node can take input from one node
- Node values can be hard coded or dynamic via N8N expressions

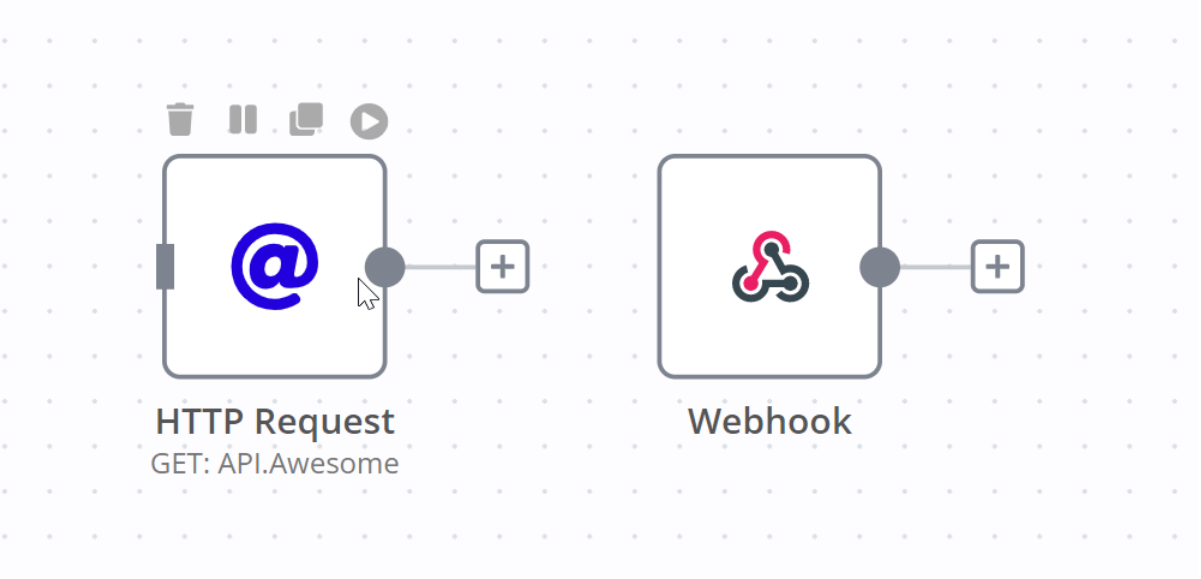


# N8N Expressions and Node Data Access

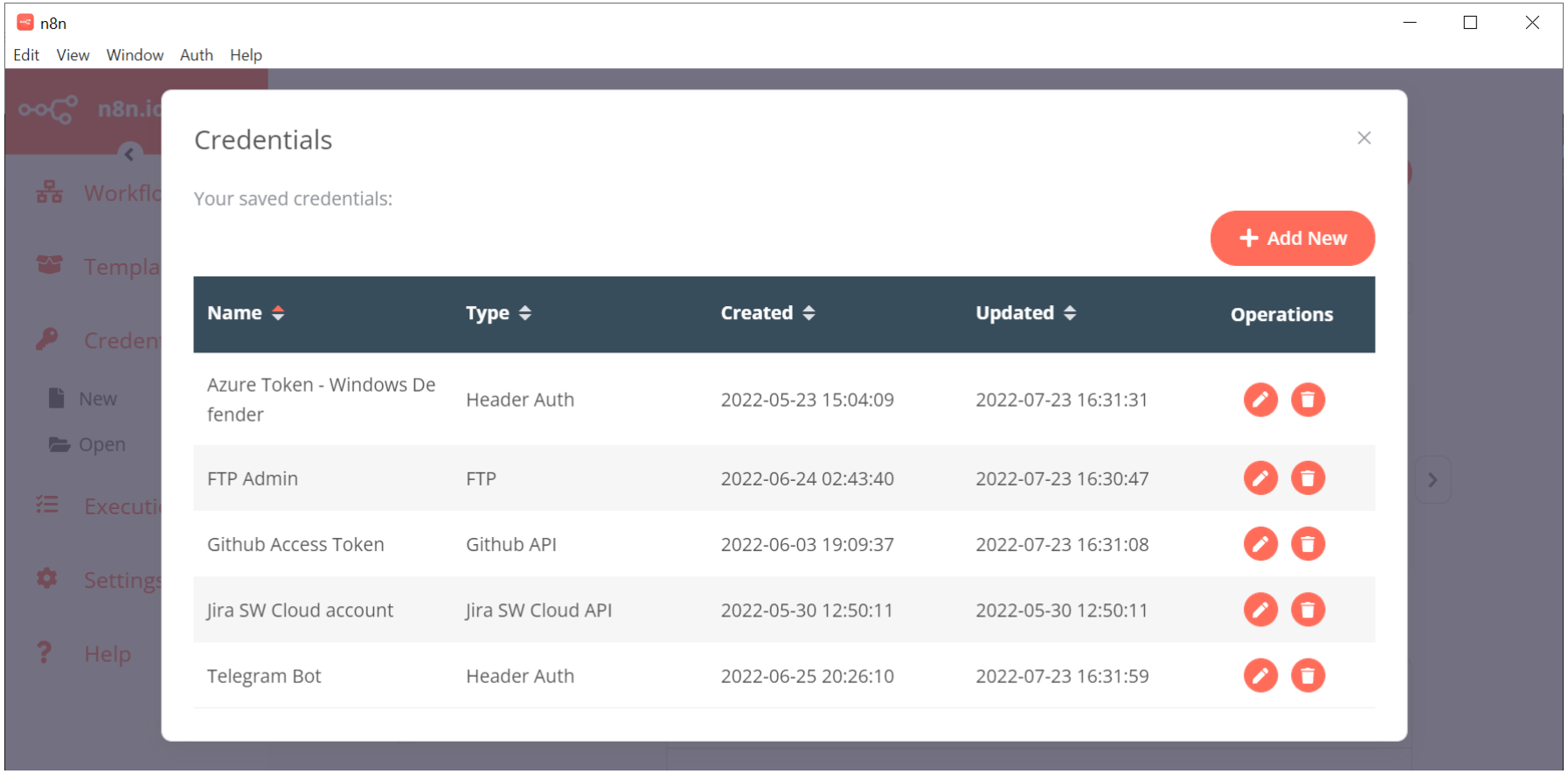
- N8N Expressions
- Data Access using $nod[“NodeName”].json
- Data Access using Copy Path Feature



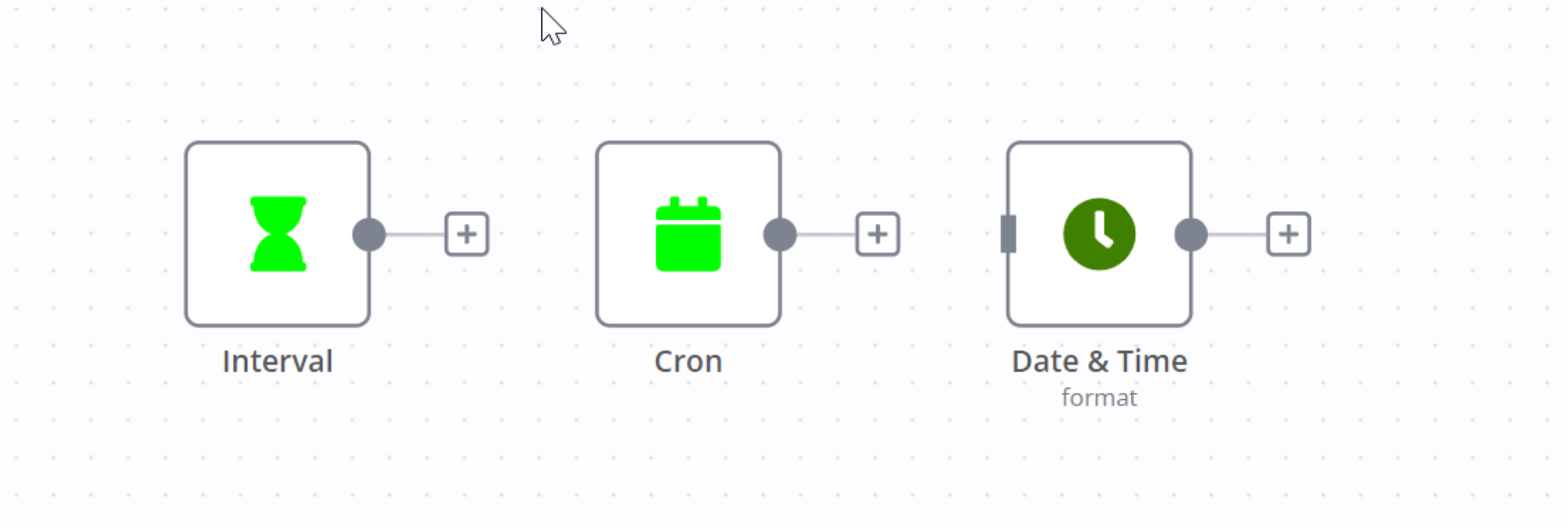
# Core Nodes



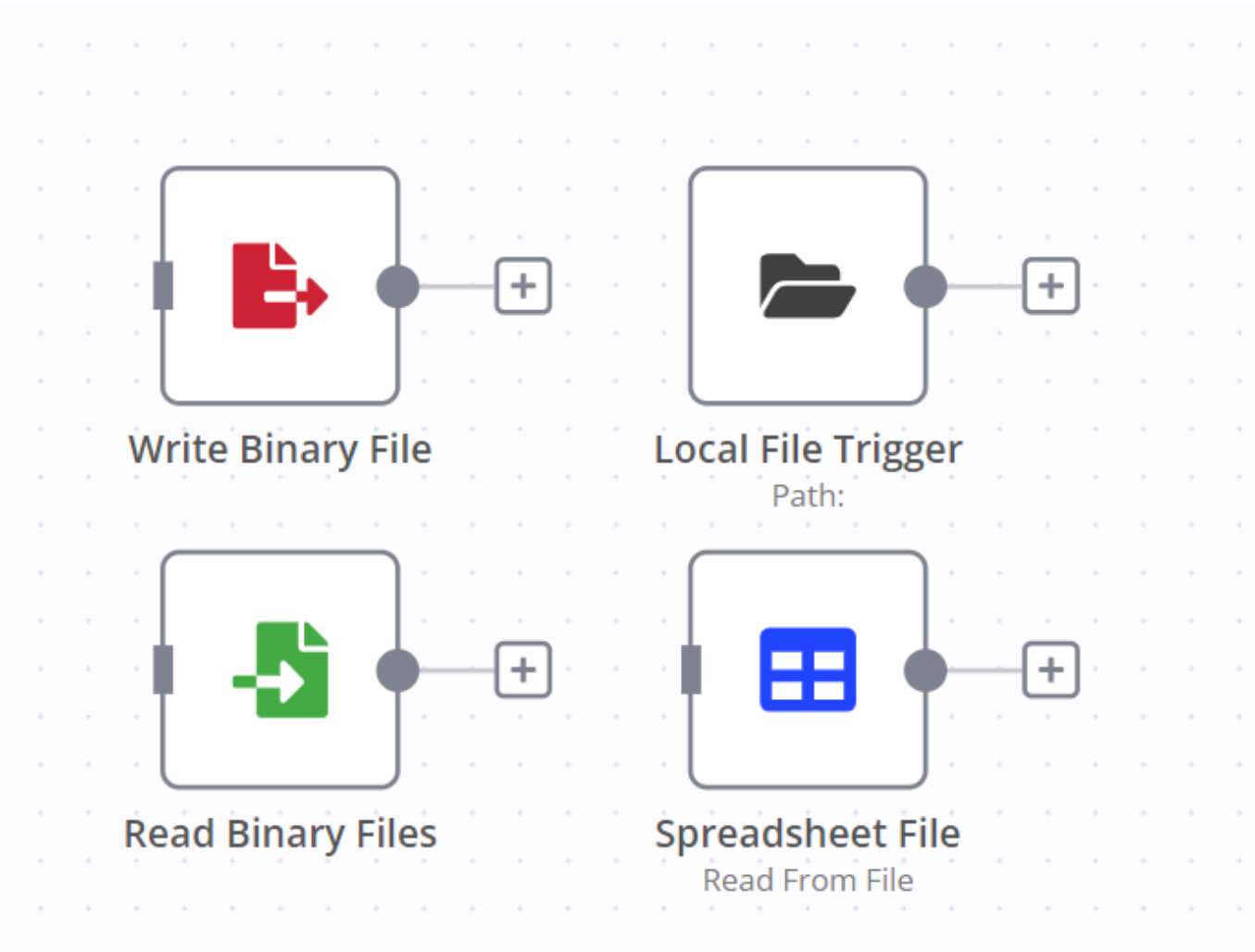
# Credential Management



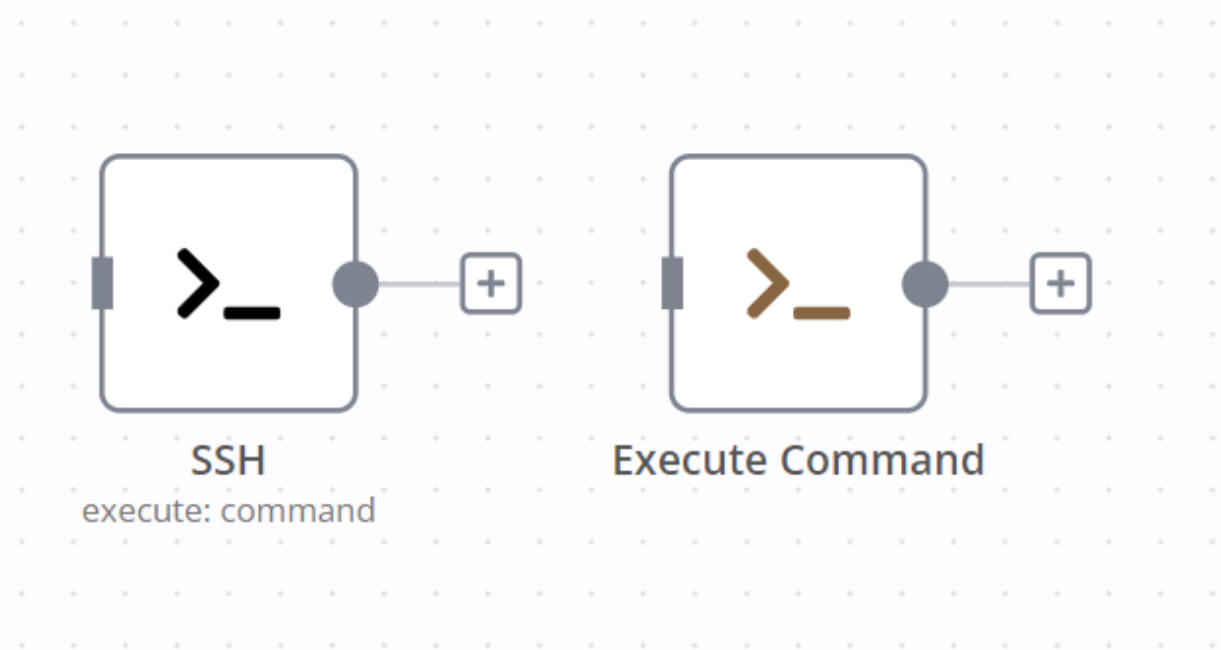
# Core Nodes



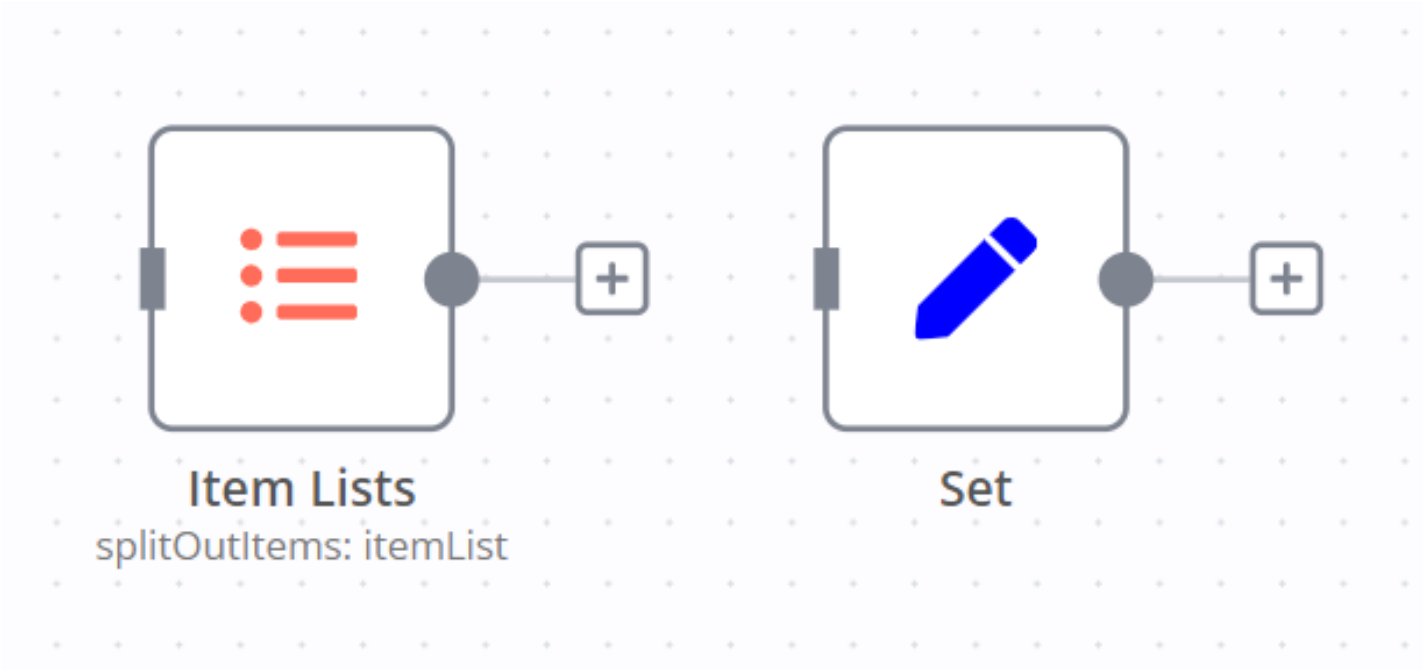
# Core Nodes


Write Binary File
Local File Trigger
Path:
Read Binary Files
Spreadsheet File
Read From File

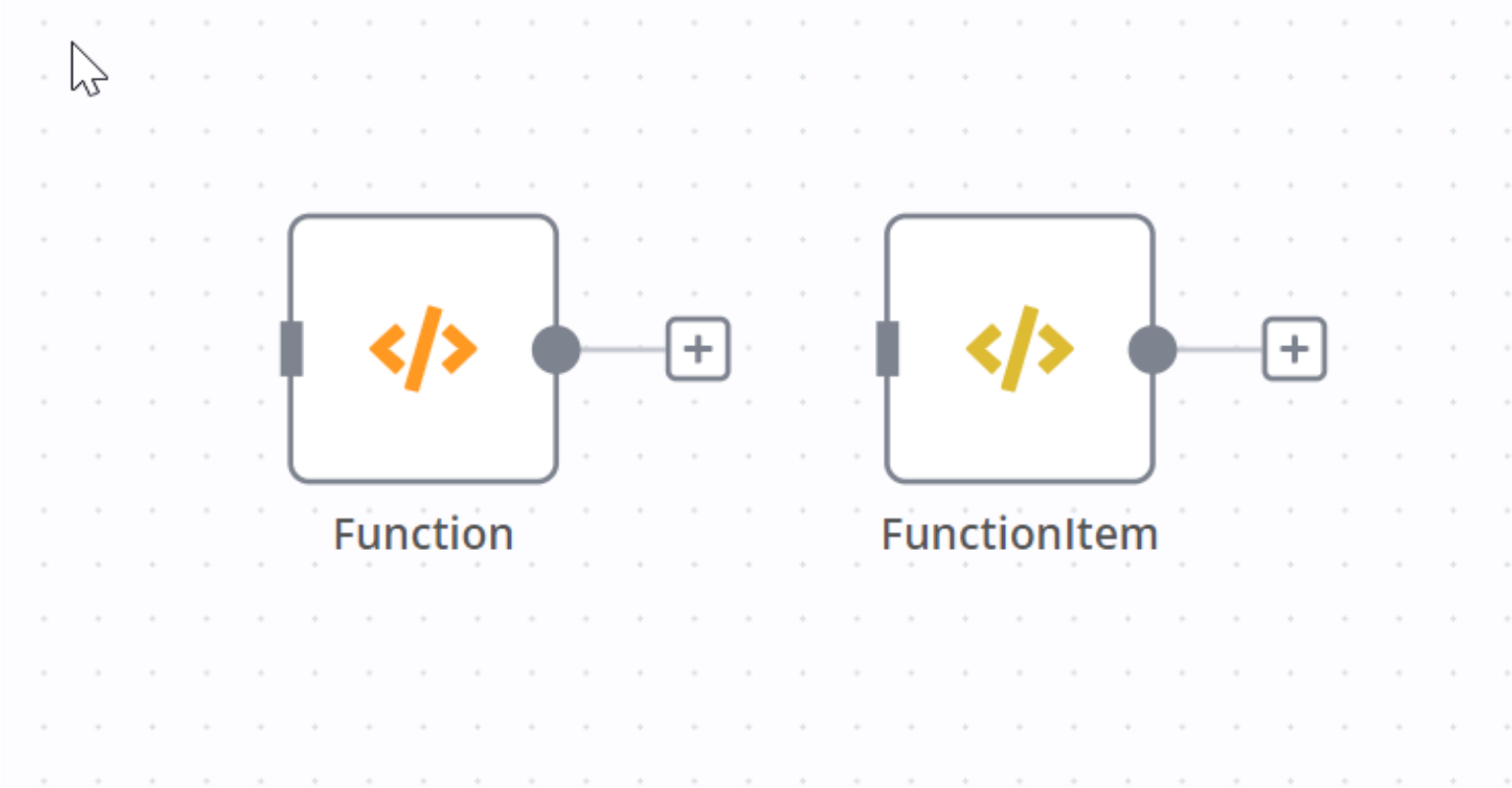
# Core Nodes



# Core Nodes

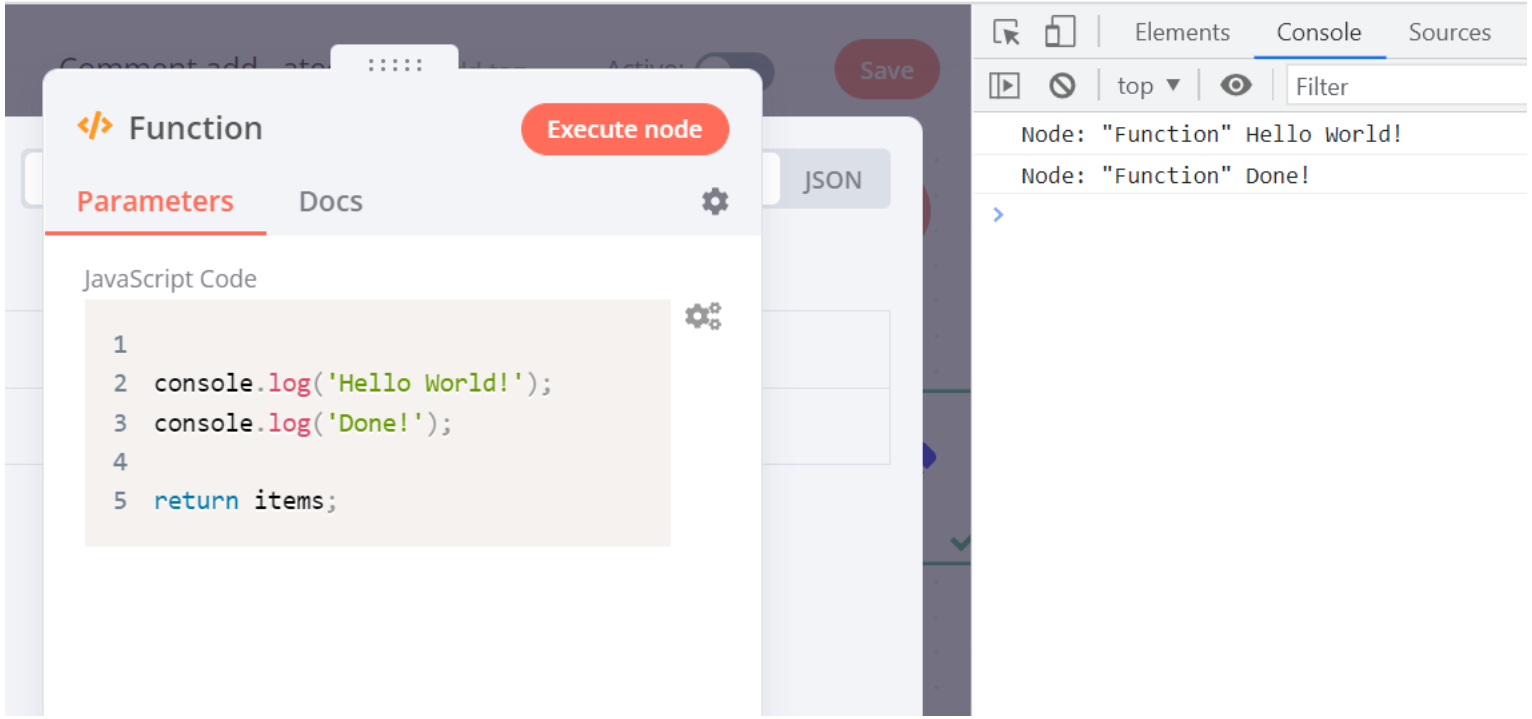


# Core Nodes

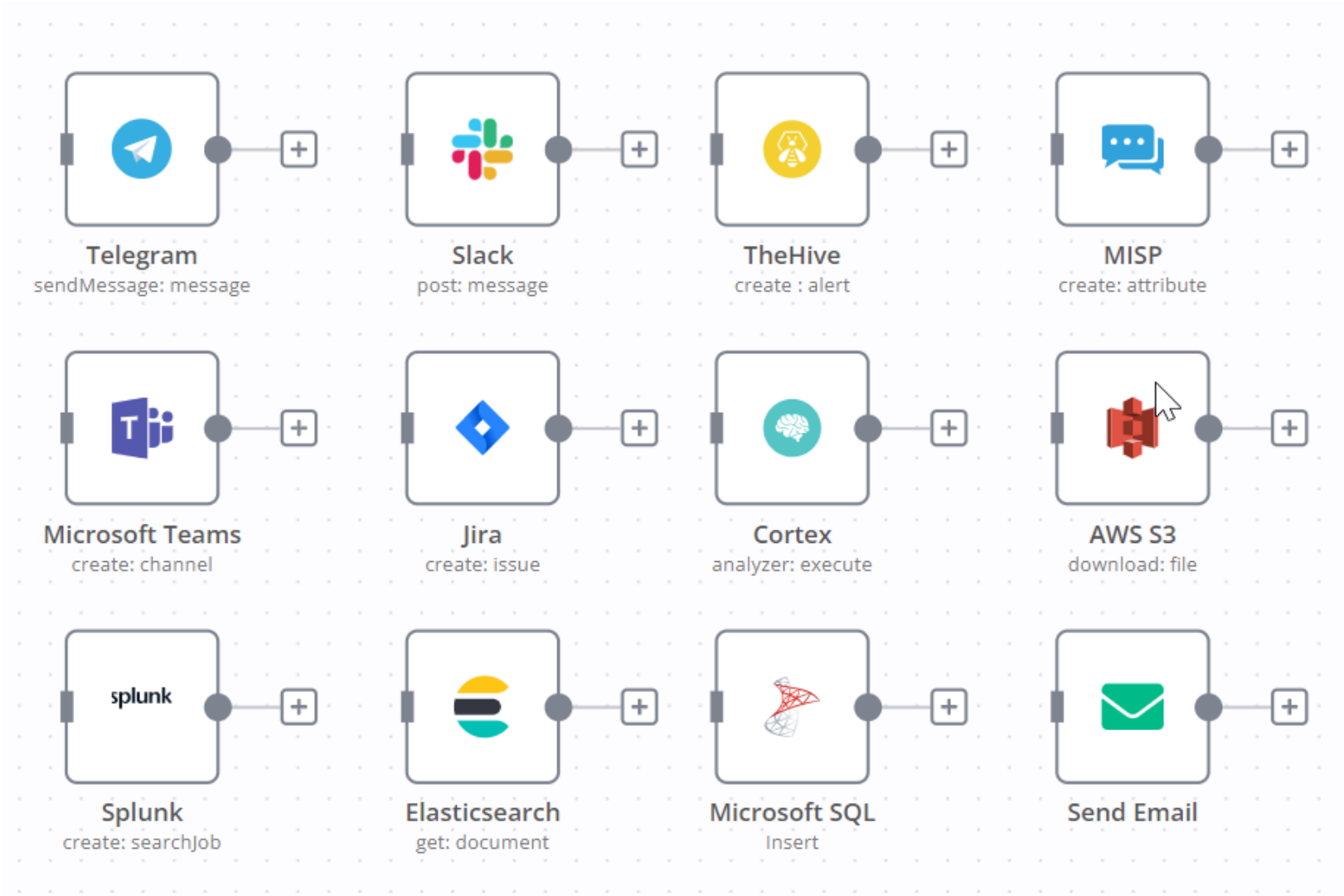


# Core Nodes

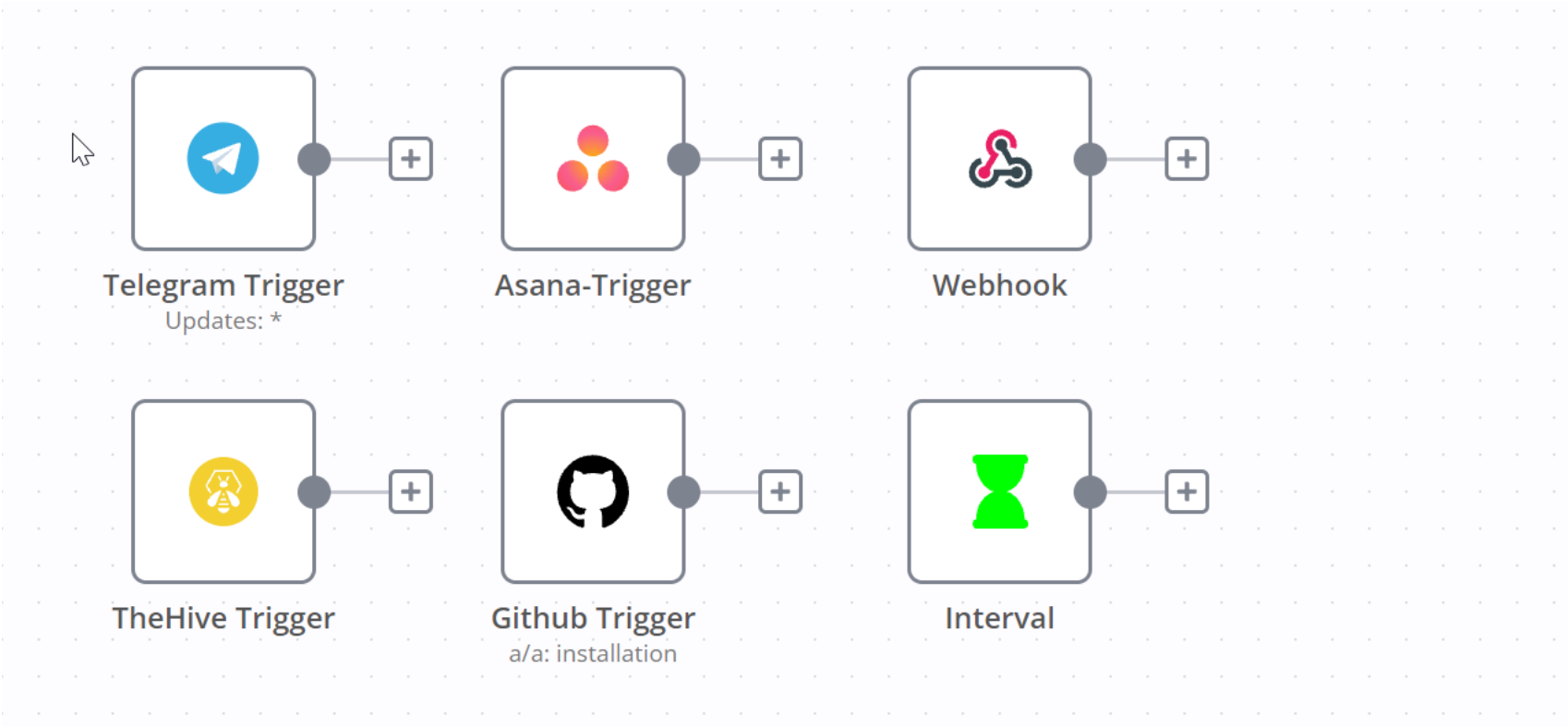
N8N can output to Browser JS console, useful for code debugging.



# Other Nodes

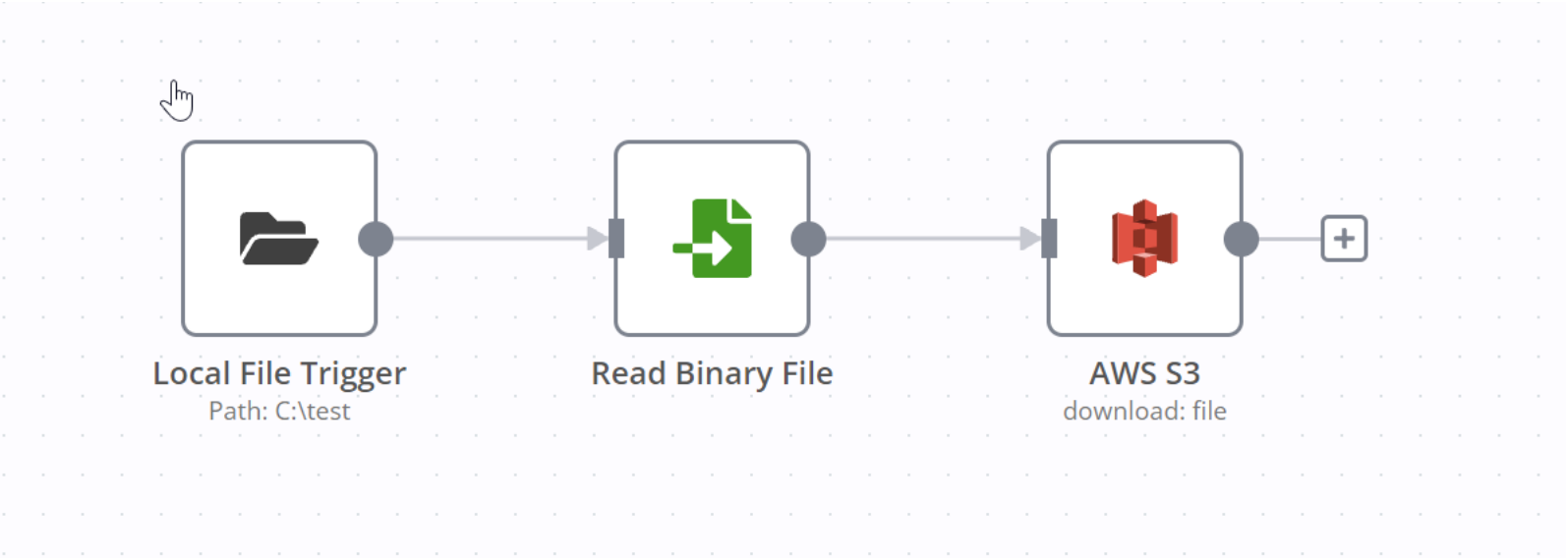

Telegram
sendMessage: message
Slack
post: message
TheHive
create : alert
MISP
create: attribute
Microsoft Teams
create: channel
Jira
create: issue
Cortex
analyzer: execute
AWS S3
download: file
splunk
Splunk
create: searchJob
Elasticsearch
get: document
Microsoft SQL
Insert
Send Email

# Triggers


Telegram Trigger
Updates: *
Asana-Trigger
Webhook
TheHive Trigger
Github Trigger
a/a: installation
Interval

# Hello Automation World

Automatically upload files to S3 Bucket



# JMESPATH

**JMESPath is a query language for JSON.**

```
locations[?state == 'WA'].name | sort(@) | {WashingtonCities: join(', ', @)}
```

```
{
  "locations": [
    {"name": "Seattle", "state": "WA"},
    {"name": "New York", "state": "NY"},
    {"name": "Bellevue", "state": "WA"}
    {"name": "Olympia", "state": "WA"}
  ]
}
```

https://jmespath.org/tutorial.html

# Basic Expressions

```
a.b.c.d
```

```
{"a": {"b": {"c": {"d": "value"}}}}
```

Result

```
"value"
```

You can use a *subexpression* to return to nested values in a JSON object:

```
a.b.c[0].d[1][0]
```

```
{"a": {
  "b": {
    "c": [
      {"d": [0, [1, 2]]},
      {"d": [3, 4]}
    ]
  }
}}
```

Result

```
1
```

You can combine identifiers, sub expressions, and index expressions to access JSON elements.

# Basic Expressions

```
a.b.c[0].d[1][0]
```

```
{"a": {
  "b": {
    "c": [
      {"d": [0, [1, 2]]},
      {"d": [3, 4]}
    ]
  }
}}
```

Result

```
1
```

You can combine identifiers, sub expressions, and index expressions to access JSON elements.

```
[5:10]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Result

```
[
  5,
  6,
  7,
  8,
  9
]
```

This slice result contains the elements 0, 1, 2, 3, and 4. The element at index 5 is not included. If we want to select the second half of the array, we can use this expression:

# Basic Expressions

```
people[*].first
```

```
{
  "people": [
    {"first": "James", "last": "d"},
    {"first": "Jacob", "last": "e"},
    {"first": "Jayden", "last": "f"},
    {"missing": "different"}
  ],
  "foo": {"bar": "baz"}
}
```

Result

```
[
  "James",
  "Jacob",
  "Jayden"
]
```

A wildcard expression creates a list projection, which is a projection over a JSON array. This is best illustrated with an example. Let's say we have a JSON document describing a people, and each array element is a JSON object that has a first, last, and age key. Suppose we wanted a list of all the first names in our list.

# Basic Expressions

```
reservations[*].instances[*].state
```

```
{
  "reservations": [
    {
      "instances": [
        {"state": "running"},
        {"state": "stopped"}
      ]
    },
    {
      "instances": [
        {"state": "terminated"},
        {"state": "running"}
      ]
    }
  ]
}
```

Result

```
[
  [
    "running",
    "stopped"
  ],
  [
    "terminated",
    "running"
  ]
]
```

More than one projection can be used in a JMESPath expression. In the case of a List/Object projection, the structure of the original document is preserved when creating projection within a projection. For example, let's take the expression reservations[*].instances[*].state.

# Basic Expressions

```
myarray[?contains(@, 'foo') == `true`]
```

```
{
  "myarray": [
    "foo",
    "foobar",
    "barfoo",
    "bar",
    "baz",
    "barbaz",
    "barfoobaz"
  ]
}
```

Result

```
[
  "foo",
  "foobar",
  "barfoo",
  "barfoobaz"
]
```

Functions can also be combined with filter expressions. In the example below, the JMESPath expressions finds all elements in myarray that contains the string foo.

# Luxon Support

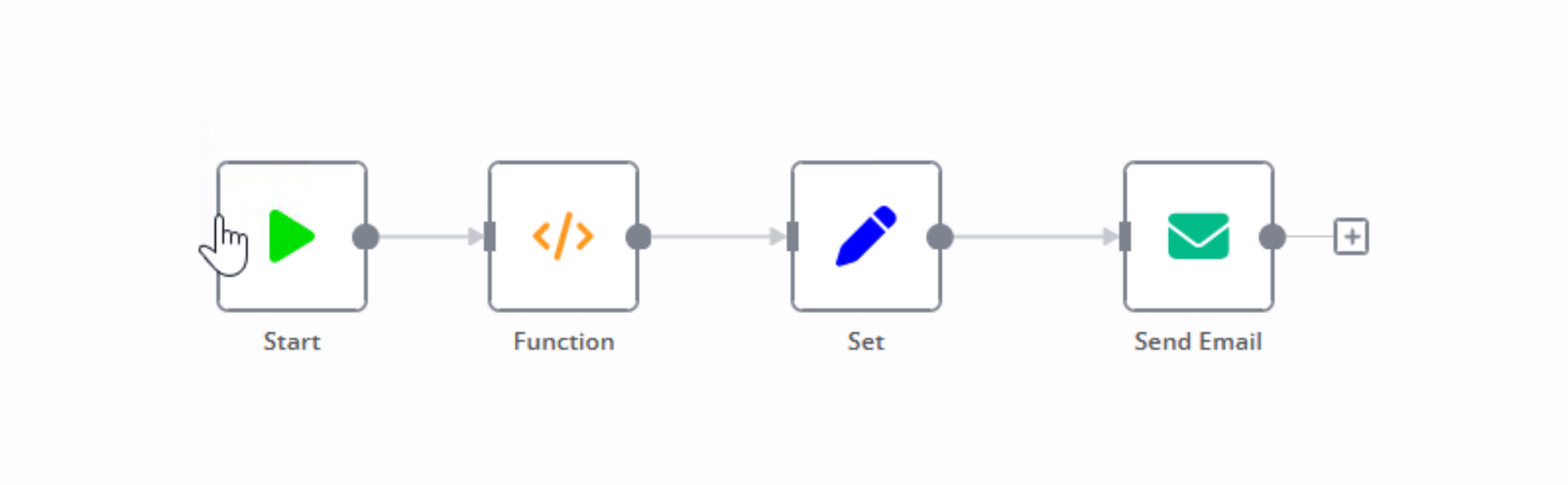
Luxon is a library for dealing with dates and times in JavaScript.

**Features**

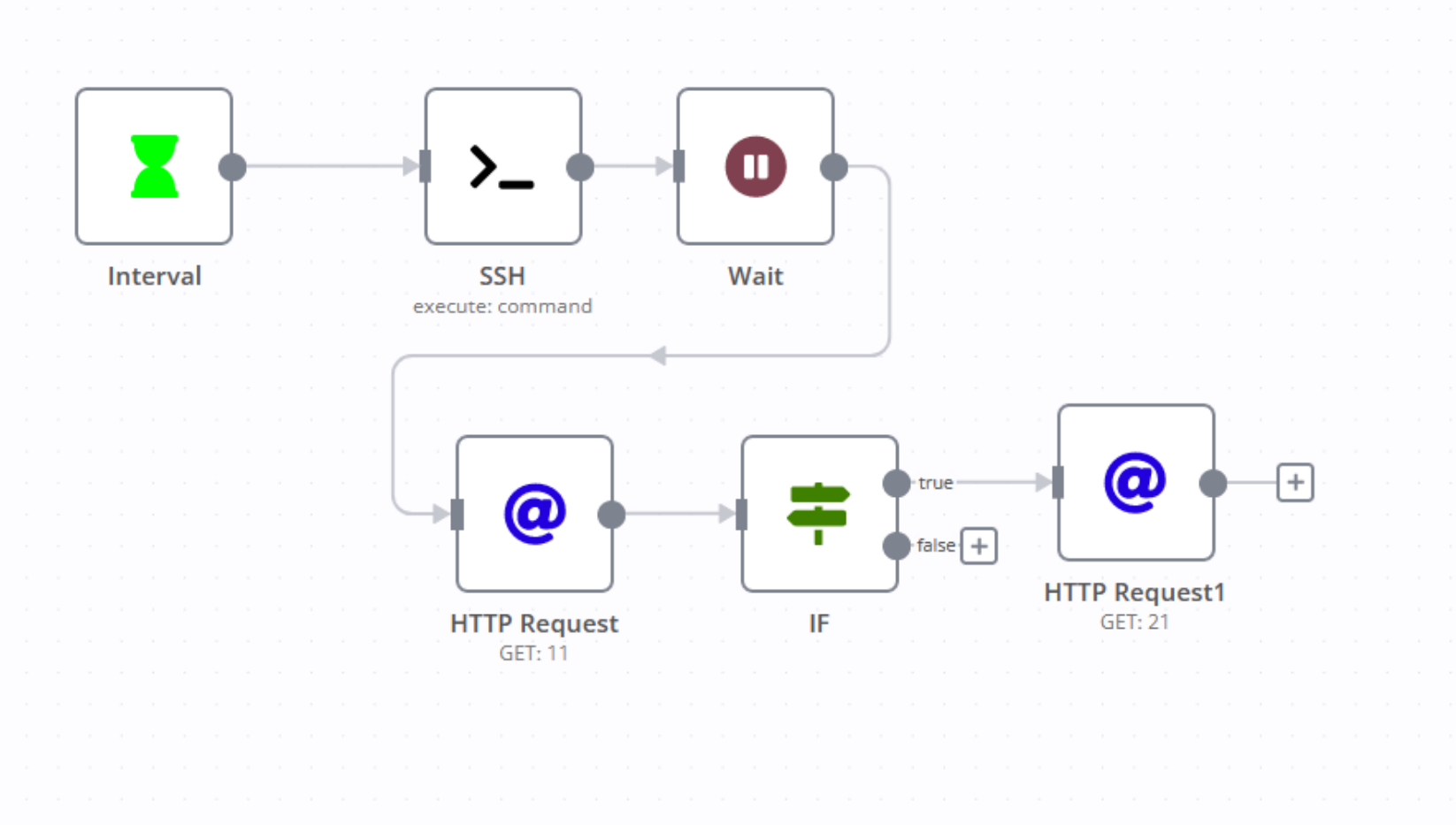
- Time Formatting
- Math Supports
- Time zones
- Calendars
- Parsing
- Durations
- Intervals

https://moment.github.io/luxon/#/

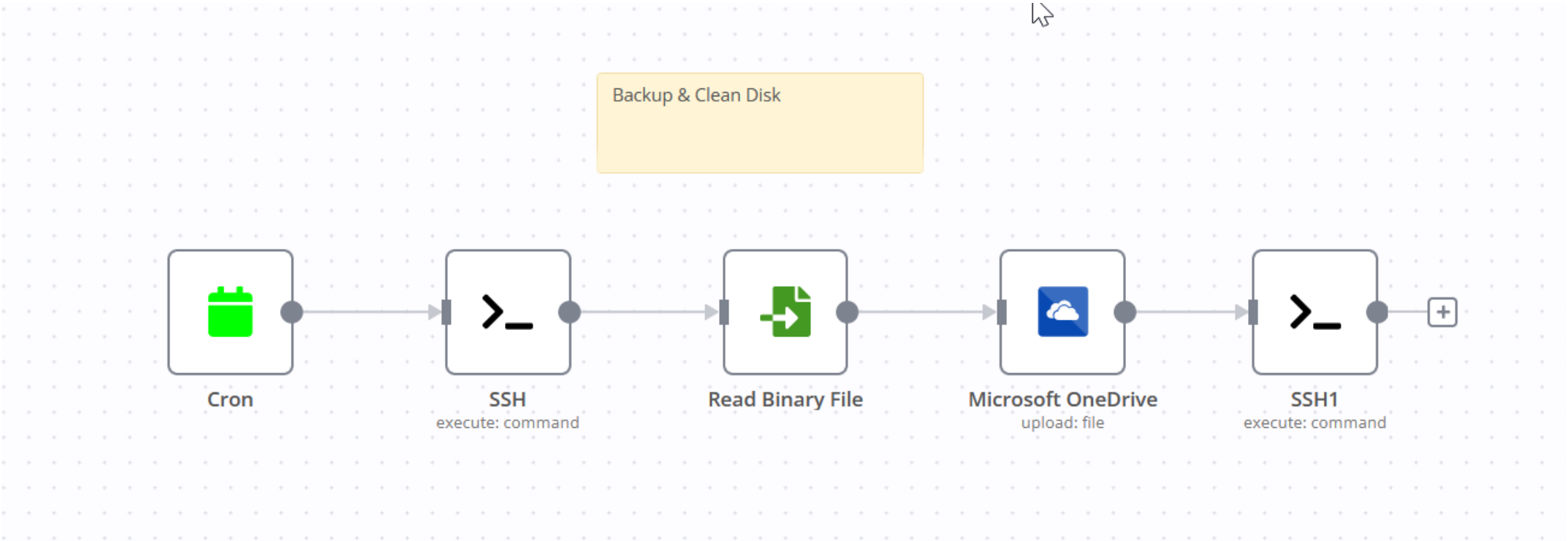
# Working With JSON



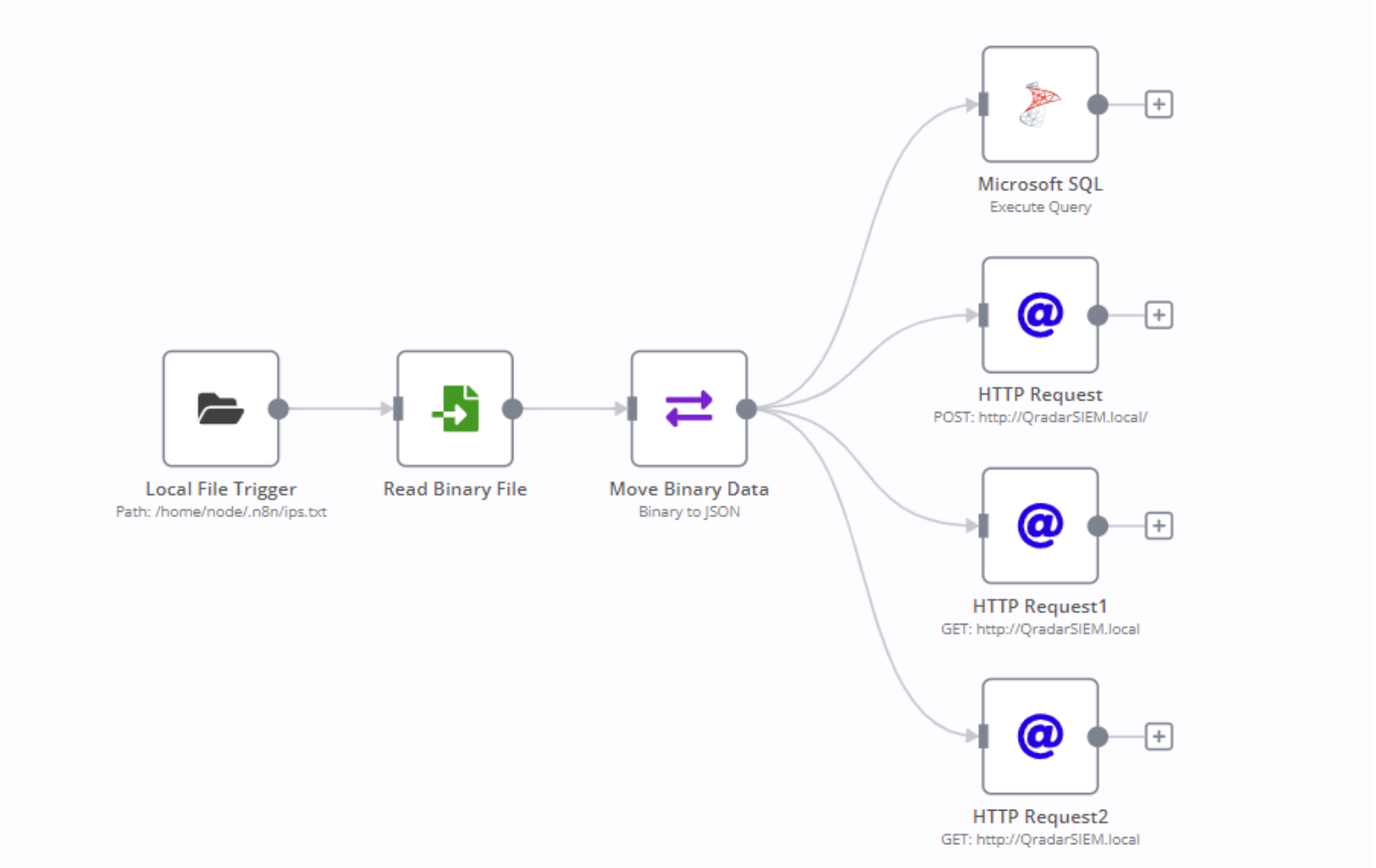
# Detection Testing



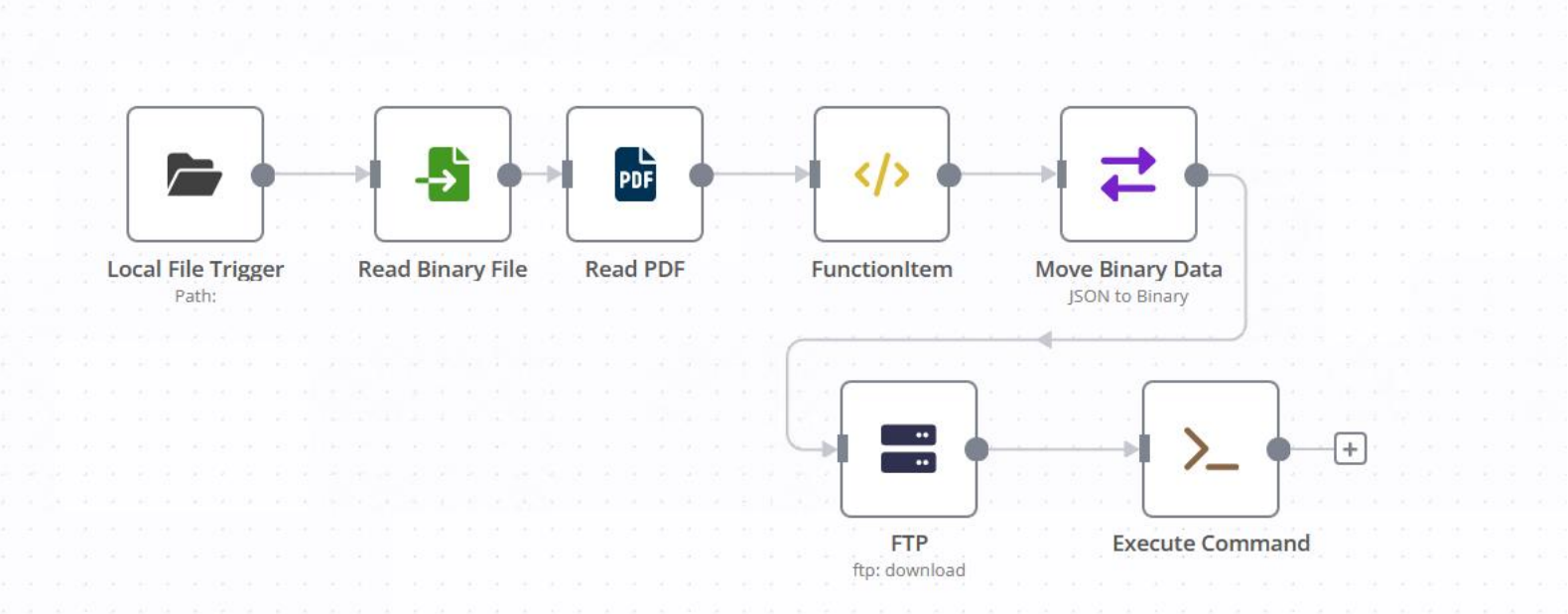
# Infrastructure Operation



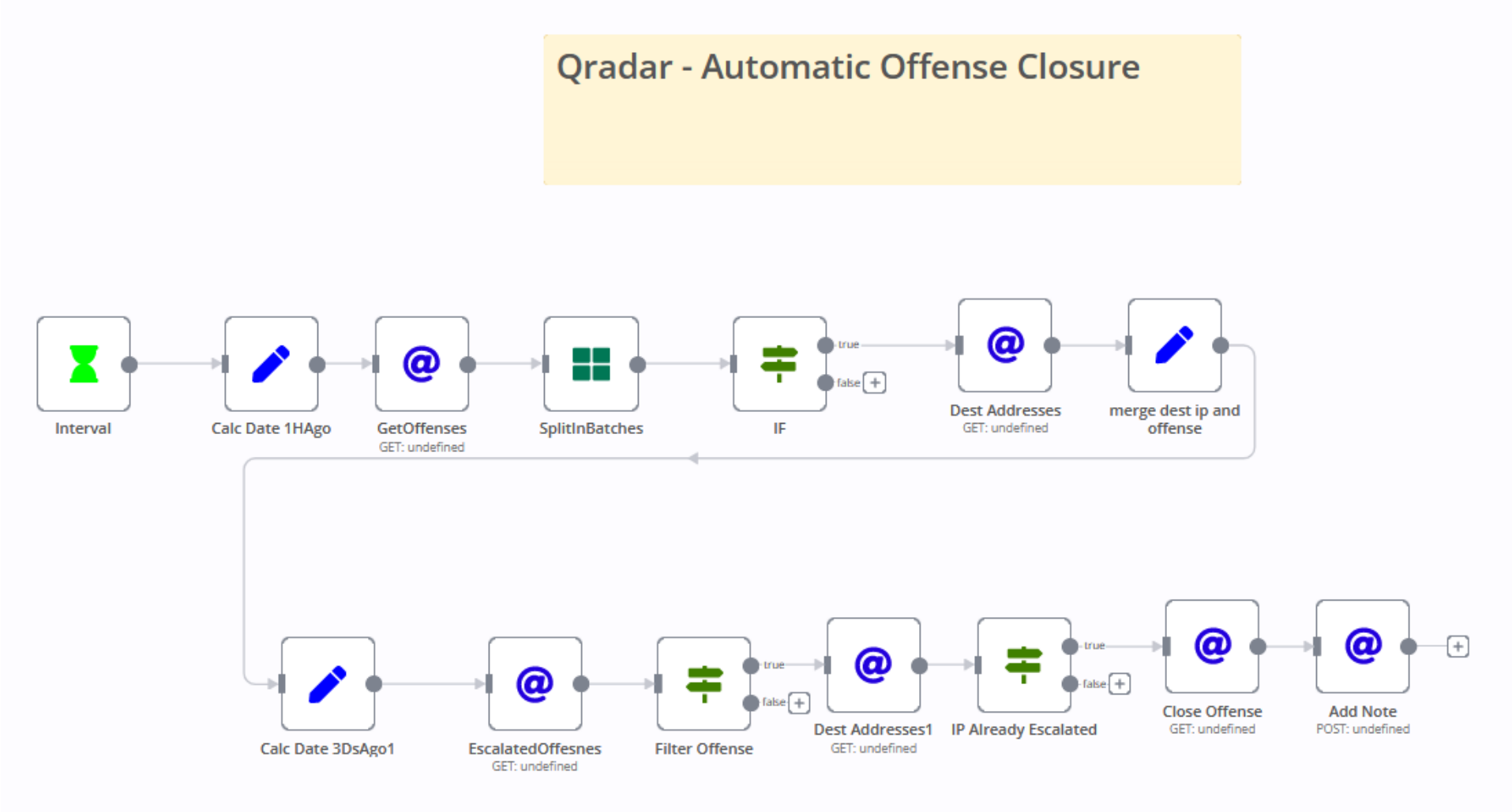
# Intel ingestion



# Automatic Yara Extraction

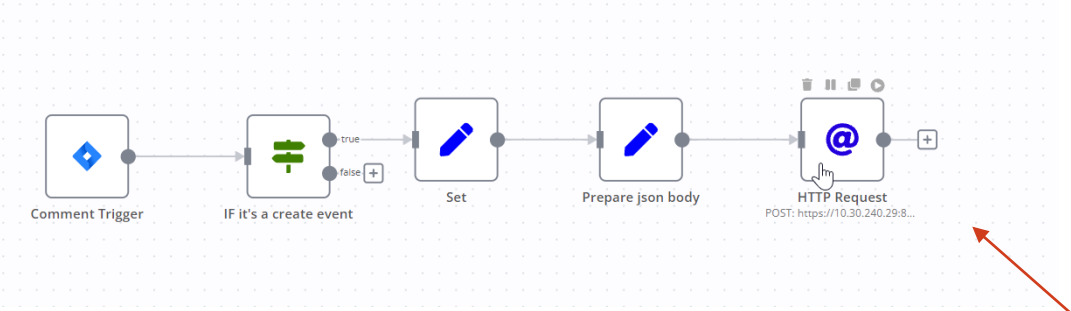


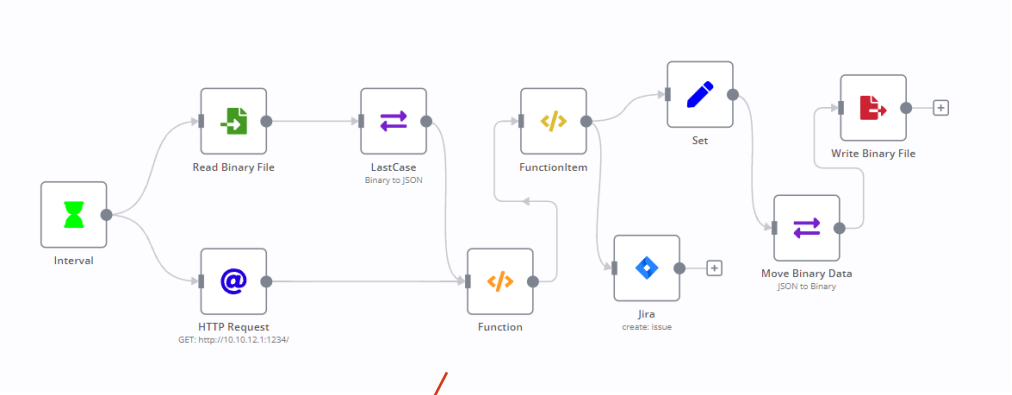
# Automated Alert Closure

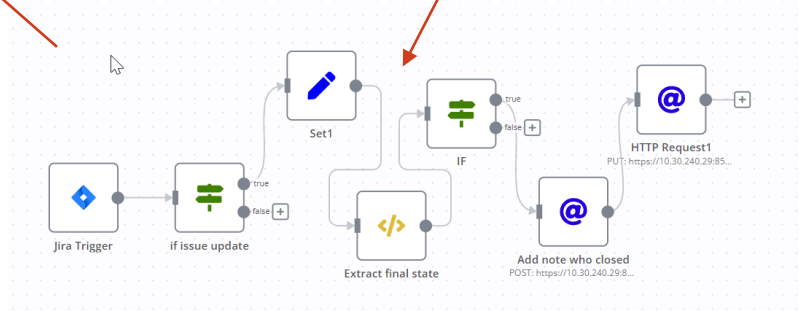


# Multiple N8N WorkFlows Automation

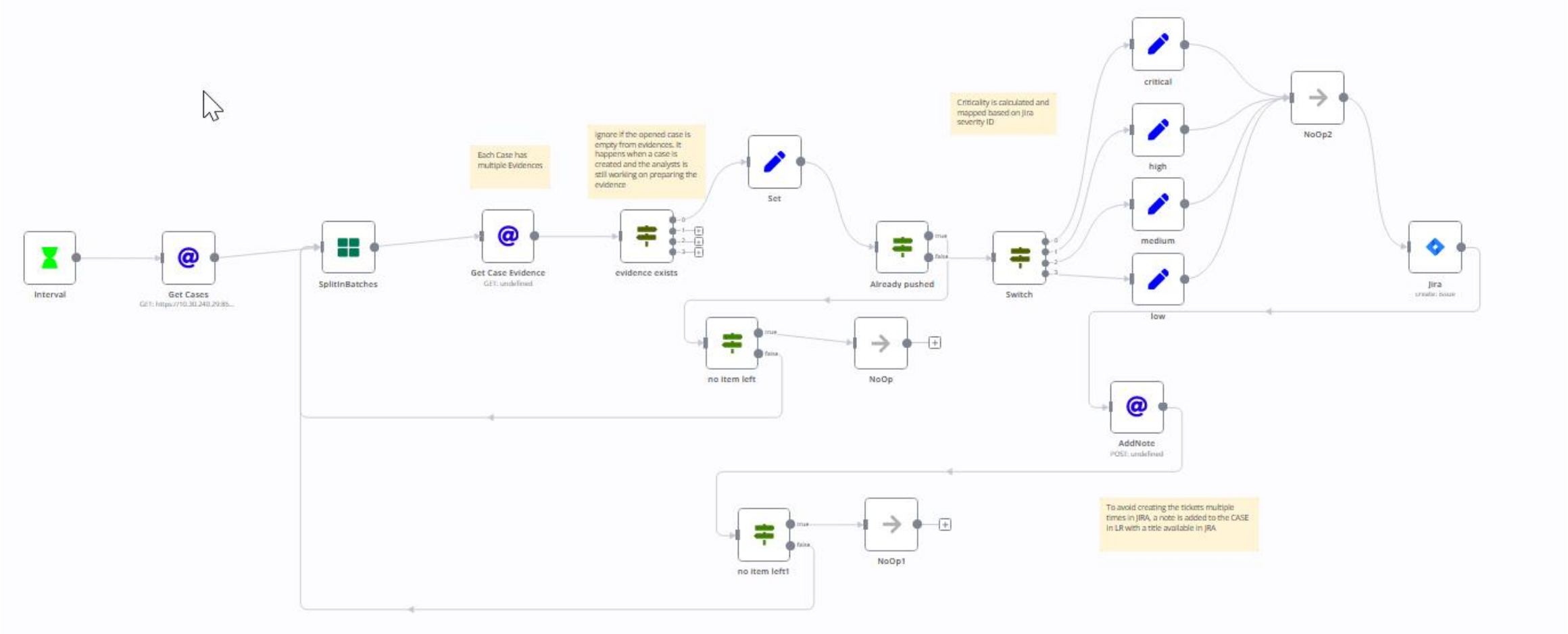
- One workflow orchestrate multiple N8N nodes to automate a task
- Multiple N8N workflows can automate entire workflow



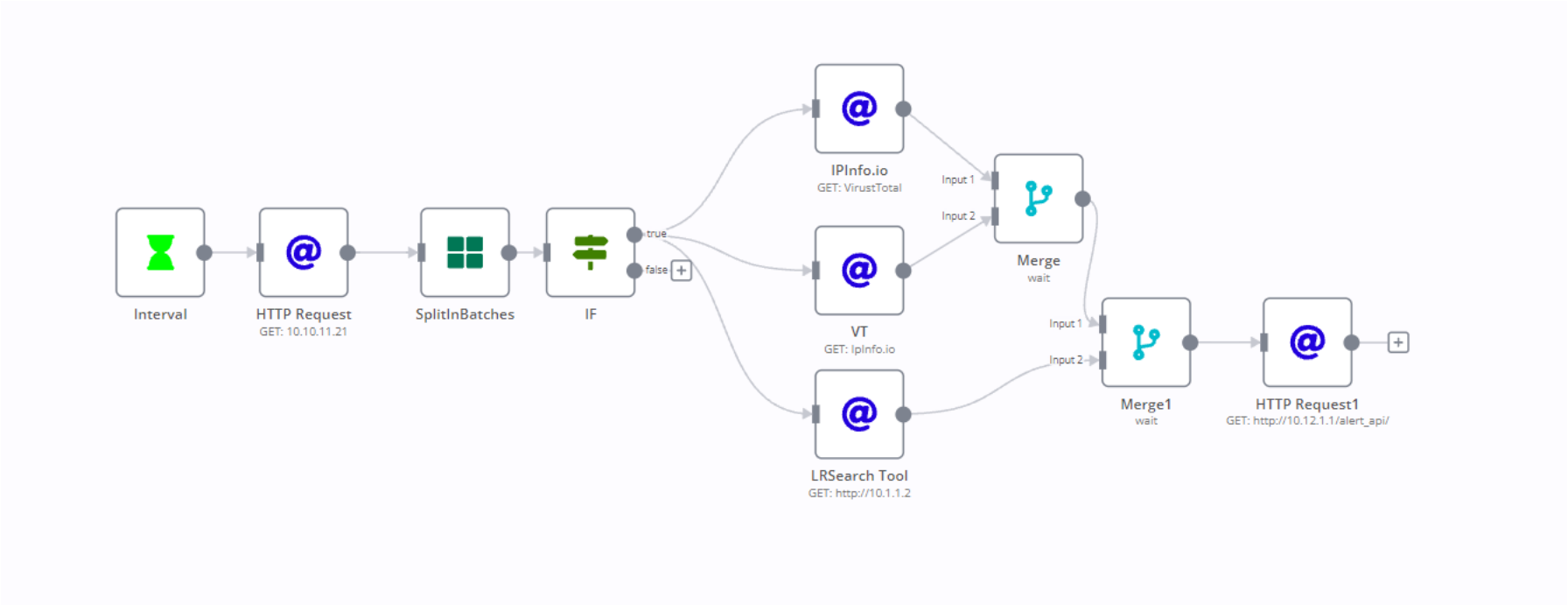



# Ticketing System Integration



# Intelligence Enrichment



# SOC Reporting