FLOWX.AI

PLATFORM OVERVIEW / Frameworks and standards

Contents

PLATFORM OVERVIEW / Frameworks and standards / Business process industry and standards / Intro to BPMN / BPMN basic concepts

Let's get into a bit more details on the main types of BPMN process elements.

Events

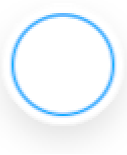
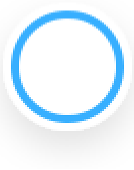
Events are **signals that something happens** – this includes the start and end of a process as well as any interaction with the process’ environment.

There are 3 types of events:

- start events
- end events
- intermediate events

Start and End events

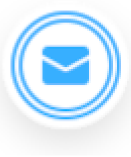
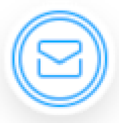
Start & End events

Start Event Icon	End Event Icon
	
event that triggers the process	event that defines the state that terminates the process

Intermediate events

Message events

- represents incoming or outgoing messages from external parties - information, email, bank transfer
- Receive Message Event - incoming message occurring during the process flow, somewhere between start and end
- Send Message Event - outgoing message

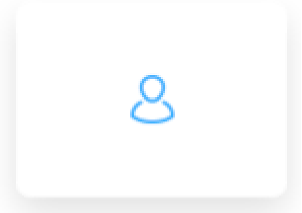
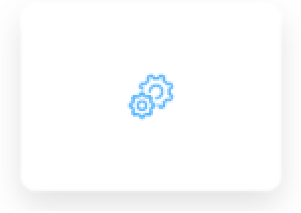
Send Message Event Icon	Receive Message Event Icon
	

Send Message Event Icon	Receive Message Event Icon
outgoing message	incoming message

Activities

Task

- it is an atomic activity within a process flow. You create a task when the activity cannot be broken down to a finer level of detail. A task can only belong to one lane.

User task	Service task
	
a task that requires the human to perform an action	a task that uses a Web service, an automated application, or other kinds of service in completing the task.

Send Task

- represents a task that sends a Message to another lane or pool. The Task is completed once the Message has been sent.

Receive Task

- indicates that the process has to wait for a message to arrive in order to continue. The Task is completed once the message has received.

User Task

- is a Task that is performed without the aid of any business process execution engine or any application. It is performed when the user performs a certain action in the application.

Service Task

- is executed by a business process engine. The task defines a script that the **process engine** can interpret. When the task begin, the engine will execute the script. The Task will be completed when the script is completed. It also provides a mechanism for a process to run a **business rule** on the process data.

BPMN Subprocesses

In BPMN, a subprocess is a compound activity that represents a collection of other tasks and subprocesses. Generally, we create BPMN diagrams to communicate processes with others. To facilitate effective communications, we really do not want to make a business process diagram too complex. By using subprocesses, you can split a complex process into multiple levels, which allows you to focus on a particular area in a single process diagram.

Gateways

Gateways allow to control as well as merge and split the **process flow**.

Exclusive gateways

In business processes, you typically need to make choices — **business decisions**. The most common type of decision is choosing **either/or**. Exclusive Gateways limit the possible outcome of a decision to a single path, and circumstances choose which one to follow.

Parallel gateways

In many cases, you want to split up the flow within your business process. For example the sales and risk departments may examine a new mortgage application at the same time. This reduces the total cycle time for a case. To express parallel flow in BPMN, you use a **parallel gateway**.

Exclusive gateway (XOR)	Parallel gateway (AND)
✕	+
• defines a decision point	• no decision making; • all outgoing branches are activated

Closing gateway

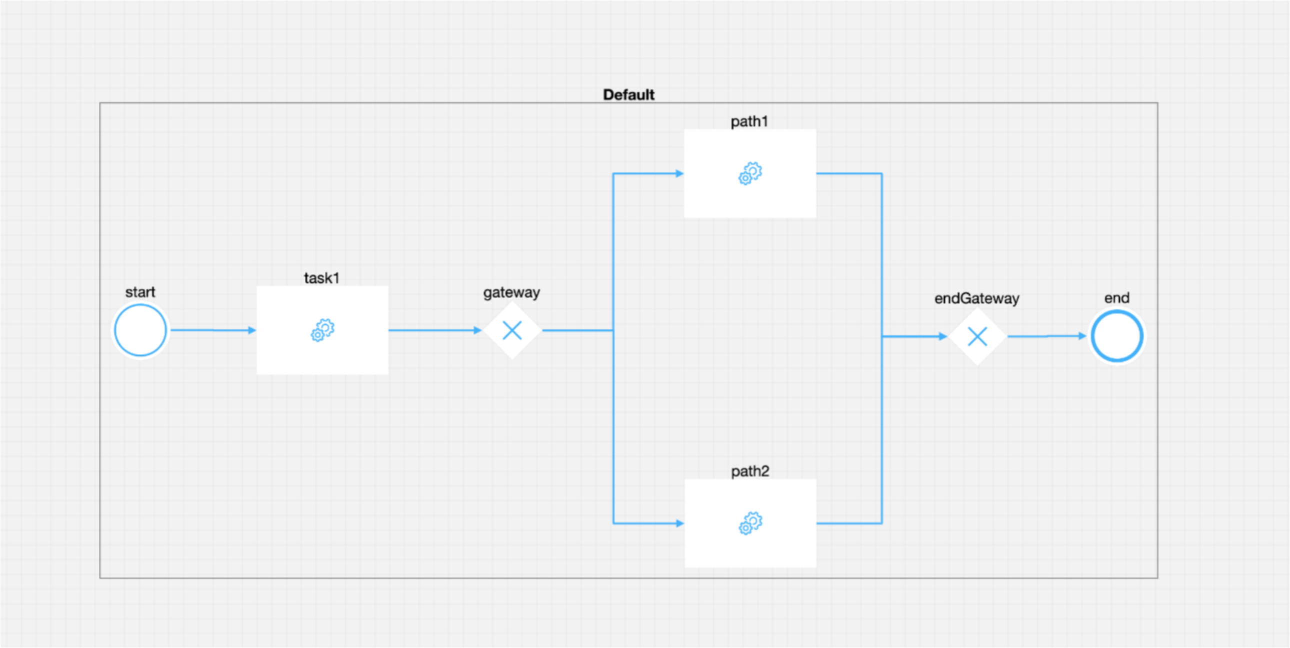
- closes gateways by connecting branches with no logic involved
- symbol used depends on the initial gateway
- parallel gateways - waits for all input tokens and merges all into one single token
- inclusive gateways

  - waits for all active inputs
  - is informed about all preceding token flows - knows the path selected and are expecting the token from these

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Business process industry and standards / Intro to DMN

As we've seen in the previous chapter, Business Process Model and Notation (BPMN) is used to define business processes as a sequence of activities. If we need to branch off different process paths, we use gateways. These have rules attached to them in order to decide on which outgoing path should the process continue on.



> ⓘ **INFO**
>
> For more information on how to define DMN gateway decisions, check the **Exclusive gateway node** section.

We needed a convenient way of specifying the

The fallback content to display on prerendering
and we picked two possible ways of writing business rules:

- defining them as DMN decisions

> ⓘ **INFO**
>
> You can now define a DMN Business Rule Action directly in
>
> The fallback content to display on prerendering
> . For more information, check the **DMN Business Rule Action** section.

- adding MVEL scripts

What is Decision Model and Notation (DMN)?

Decision Model and Notation (or DMN) is a graphical language that is used to specify business decisions. DMN acts as a translator, converting the code behind complex decision-making into easily readable diagrams.

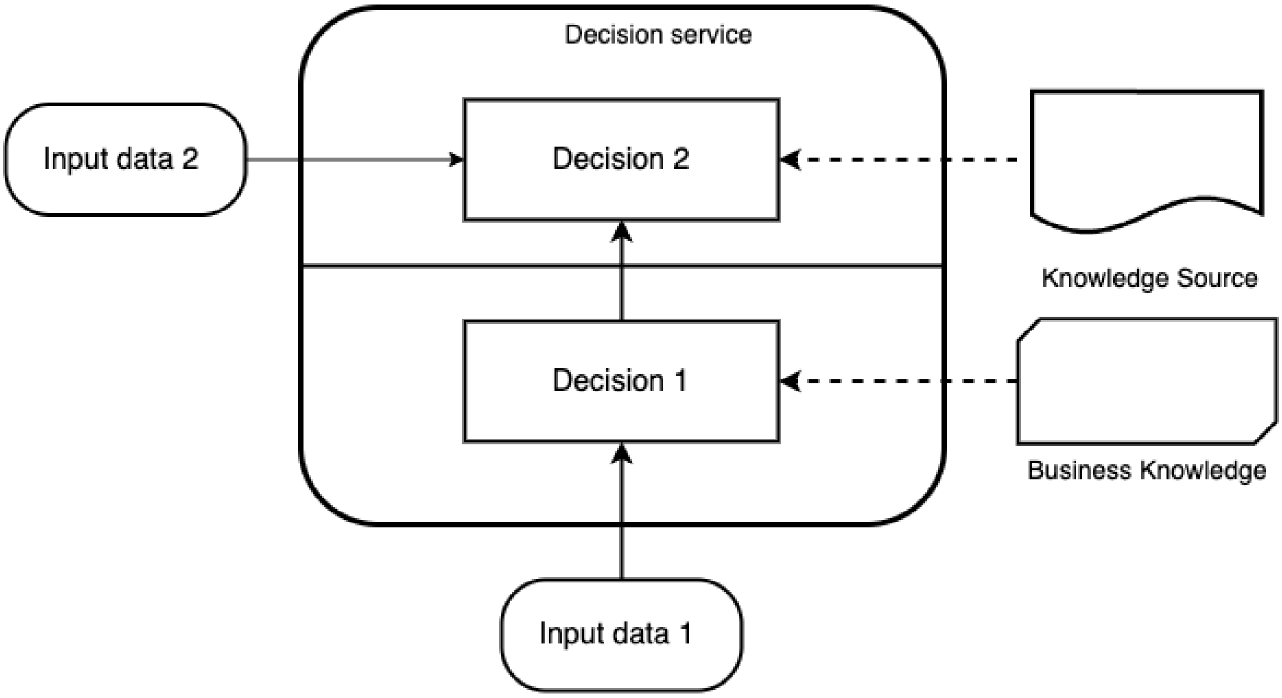
The Business Process Model and Notation is used to create the majority of process models **(BPMN)**. The DMN standard was developed to complement BPMN by providing a mechanism for modeling decision-making represented by a Task within a process model. DMN does not have to be used in conjunction with BPMN, but it is highly compatible.

> **DANGER**
>
> FLOWX.AI supports DMN 1.3 version.

DMN Elements

There are 4 basic elements of the **Decision Model and Notation**:

- Decision
- Business Knowledge Model
- Input Data
- Knowledge Source



Decision

It’s the center point of a DMN diagram and it symbolizes the action that determines as output the result of a decision.

Decision service

A decision service is a high-level decision with well-defined inputs that is made available as a service for invocation. An external application or business process can call the decision service (BPMN).

Business Knowledge Model

It portrays a specific knowledge within the business. It stores the origin of the information. Decisions that have the same logic but depend on different sub-input data or sub-decisions use business knowledge models to determine which procedure to follow.

Example: a decision, rule, or standard table.

Input Data

This is the information used as an input to the normal decision. It's the variable that configures the result. Input data usually includes business-level concepts or objects relevant to the business.

Example: Entering a customer's tax number and the amount requested in a credit assessment decision.

Knowledge Source

It's a source of knowledge that conveys a kind of legitimacy to the business.

Example: policy, legislation, rules.

DMN Decision Table

A decision table represents decision logic which can be depicted as a table in Decision Model and Notation. It consists of inputs, outputs, rules, and hit policy.

Decision table elements	
Inputs	A decision table can have one or more input clauses, that represent the attributes on which the rule should be applied.

Decision table elements	
Outputs	Each entry with values for the input clause needs to be associated with output clauses. The output represents the result that we set if the rules applied to the input are met.
Rules	Each rule contains input and output entries. The input entries are the condition and the output entries are the conclusion of the rule. If each input entry (condition) is satisfied, then the rule is satisfied and the decision result contains the output entries (conclusion) of this rule.
Hit policy	The hit policy specifies what the result of the decision table is in cases of overlapping rules, for example, when more than one rule matches the input data.

Hit Policy examples

Unique **First** **Priority** **Any** **Rule order** **Collect order**

- unique result
- only one rule will match, or no rule

DMN Model

DMN defines an XML schema that allows DMN models to be used across multiple DMN authoring platforms.

You can use this XML example with FLOWX Designer, adding it to a Business Rule Action - using an MVEL script. Then you can switch to DMN if you need to generate a graphical representation of the model.

Using DMN with FLOWX Designer

As mentioned previously, DMN can be used with FLOWX Designer for the following scenarios:

- For defining gateway decisions, using exclusive gateways.
- For defining business rules actions attached to a task node.

In depth docs

» DMN Documentation

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Business process industry and standards / Intro to MVEL

We can also **specify the business rules logic using MVEL scripts**. As opposed to DMN, with MVEL you can create complex business rules with multiple parameters and sub-calculations.

What is MVEL?

MVFLEX Expression Language (MVEL) is an expression language with a syntax similar to the Java programming language. This makes it relatively easy to use in order to define more complex business rules and that cannot be defined using DMN.

The runtime allows MVEL expressions to be executed either interpretively, or through a pre-compilation process with support for runtime byte-code generation to remove overhead. We pre-compile most of the MVEL code in order to make sure the process flow advances as fast as possible.

Example

```
if( input.get("user.credit_score") >= 700 ) {

output.setNextNodeName("TASK_SET_CREDIT_CARD_TYPE_PREMIUM");
} else {

output.setNextNodeName("TASK_SET_CREDIT_CARD_TYPE_STANDARD");
}
```

In depth docs

» MVEL Documentation

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to Elasticsearch

Elasticsearch itself is not inherently event-driven, it can be integrated into event-driven architectures or workflows. External components or frameworks detect and trigger events, and Elasticsearch is utilized to efficiently index and make the event data searchable. This integration allows event-driven systems to leverage Elasticsearch's powerful search and analytics capabilities for real-time processing and retrieval of event data.

What is Elasticsearch?

Elasticsearch is a powerful and highly scalable open-source search and analytics engine built on top of the Apache Lucene library. It is designed to handle a wide range of data types and is particularly well-suited for real-time search and data analysis use cases. Elasticsearch provides a distributed, document-oriented architecture, making it capable of handling large volumes of structured, semi-structured, and unstructured data.

How it works?

At its core, Elasticsearch operates as a distributed search engine, allowing you to store, search, and retrieve large amounts of data in near real-time. It uses a schema-less JSON-based document model, where data is organized into indices, which can be thought of as databases. Within an index, documents are stored, indexed, and made searchable based on their fields. Elasticsearch also provides powerful querying capabilities, allowing you to perform complex searches, filter data, and aggregate results.

Why it is useful?

One of the key features of Elasticsearch is its distributed nature. It supports automatic data sharding, replication, and node clustering, which enables it to handle massive amounts of data across multiple servers or nodes. This distributed architecture provides high availability and fault tolerance, ensuring that data remains accessible even in the event of hardware failures or network issues.

Elasticsearch integrates with various programming languages and frameworks through its comprehensive RESTful API. It also provides official clients for popular languages like Java, Python, and JavaScript, making it easy to interact with the search engine in your preferred development environment.

Indexing & sharding

Indexing

Indexing refers to the process of adding, updating, or deleting documents in Elasticsearch. It involves taking data, typically in JSON format, and transforming it into indexed documents within an index. Each document represents a data record and contains fields with corresponding values. Elasticsearch uses an inverted index data structure to efficiently map terms or keywords to the documents containing those terms. This enables fast full-text search capabilities and retrieval of relevant documents.

Sharding

Sharding, on the other hand, is the practice of dividing index data into multiple smaller subsets called shards. Each shard is an independent, self-contained index that holds a portion of the data. By distributing data across multiple shards, Elasticsearch achieves horizontal scalability and improved performance. Sharding allows Elasticsearch to handle large amounts of data by parallelizing search and indexing operations across multiple nodes or servers.

Shards can be configured as primary or replica shards. Primary shards contain the original data, while replica shards are exact copies of the primary shards, providing redundancy and high availability. By having multiple replicas, Elasticsearch ensures data durability and fault tolerance. Replicas also enable parallel search operations, increasing search throughput.

Sharding offers several advantages. It allows data to be distributed across multiple nodes, enabling parallel processing and faster search operations. It also provides fault tolerance, as data is replicated across multiple shards. Additionally, sharding allows Elasticsearch to scale horizontally by adding more nodes and distributing the data across them.

The number of shards and their allocation can be determined during index creation or modified later. It is important to consider factors such as the size of the dataset, hardware resources, and search performance requirements when deciding on the number of shards.

For more details, check Elasticsearch documentation:

» Elasticsearch

Leveraging Elasticsearch for advanced indexing with FLOWX.AI

The integration between FLOWX.AI and Elasticsearch involves the indexing of specific keys or data from the

The fallback content to display on prerendering
to
The fallback content to display on prerendering
using Elasticsearch. This indexing process is initiated by the
The fallback content to display on prerendering
in a synchronous manner, sending the data to Elasticsearch. The data is then indexed or updated in the "process_instance" index.

To ensure effective indexing of process instances' details, a crucial step involves defining a mapping that specifies how Elasticsearch should index the received messages. This mapping is essential as the process instances' details often have specific formats. The process-engine takes care of this by automatically creating an index template during startup if it doesn't already exist. The index template acts as a blueprint, providing Elasticsearch with the necessary instructions on how to index and organize the incoming data accurately. By establishing and maintaining an appropriate index template, the integration between FLOWX.AI and Elasticsearch can seamlessly index and retrieve process instance information in a structured manner.

Kafka transport strategy

The fallback content to display on prerendering
transport strategy implies process-engine sending messages to a Kafka topic whenever there is data from a process instance to be indexed. Kafka Connect is then configured to read these messages from the topic and forward them to Elasticsearch for indexing.
This approach offers benefits such as fire-and-forget communication, where the process-engine no longer needs to spend time handling indexing requests. By decoupling the process-engine from the indexing process and leveraging Kafka as a messaging system, the overall system becomes more efficient and scalable. The process-engine can focus on its core responsibilities, while Kafka Connect takes care of transferring the messages to Elasticsearch for indexing.

To optimize indexing response time, Elasticsearch utilizes multiple indices created dynamically by the Kafka Connect connector. The creation of indices is based on the timestamp of the messages received in the Kafka topic. The frequency of index creation, such as per minute, hour, week, or month, is determined by the timestamp format configuration of the Kafka connector.

It's important to note that the timestamp used for indexing is the process instance's start date. This means that subsequent updates received for the same object will be directed to the original index for that process instance. To ensure proper identification and indexing, it is crucial that the timestamp of the message in the Kafka topic corresponds to the process instance's start date, while the key of the message aligns with the process instance's UUID. These two elements serve as unique identifiers for determining the index in which a process instance object was originally indexed.

For more details on how to configure process instance indexing through Kakfa transport, check the following section:

» Configuring elasticsearch indexing

» Configuration guidelines

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to Kafka concepts

What is Kafka?

Apache Kafka is an open-source distributed event streaming platform that can handle a high volume of data and enables you to pass messages from one end-point to another.

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives a guarantee for fault tolerance in the presence of machine failures. It can handle a large number of diverse consumers. Kafka is very fast, and performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from a page cache to a network socket.

Benefits of using Kafka

- **Reliability** − Kafka is distributed, partitioned, replicated, and fault tolerant
- **Scalability** − Kafka messaging system scales easily without downtime
- **Durability** − Kafka uses Distributed commit log which means messages persist on disk as fast as possible
- **Performance** − Kafka has high throughput for both publishing and subscribing messages. It maintains a stable performance even though many TB of messages are stored.

Key Kafka concepts

Events

Kafka encourages you to see the world as sequences of events, which it models as key-value pairs. The key and the value have some kind of structure, usually represented in your language’s type system, but fundamentally they can be anything. Events are immutable, as it is (sometimes tragically) impossible to change the past.

Topics

Because the world is filled with so many events, Kafka gives us a means to organize them and keep them in order: topics. A topic is an ordered log of events. When an external system writes an event to Kafka, it is appended to the end of a topic.

In FLOWX.AI, Kafka handles all communication between the FLOWX Engine and external plugins and integrations. It is also used for notifying running process instances when certain events occur. More information about KAFKA configuration on the section below:

» FLOWX engine setup guide

Producer

A producer is an external application that writes messages to a Kafka cluster, communicating with the cluster using Kafka's network protocol.

Consumer

The consumer is an external application that reads messages from Kafka topics and does some work with them, like filtering, aggregating, or enriching them with other information sources.

» How to create a Kafka producer

» How to create a Kafka consumer

In-depth docs

» Kafka documentation

» How Kafka works

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to Kubernetes

What is Kubernetes?

Kubernetes is an open-source container orchestration platform that automates many of the manual processes involved in containerized application deployment, management, and scaling.

The purpose of Kubernetes is to orchestrate containerized applications to run on a cluster of hosts. **Containerization** enables you to deploy multiple applications using the same operating system on a single virtual machine or server.

Kubernetes, as an open platform, enables you to build applications using your preferred programming language, operating system, libraries, or messaging bus. To schedule and deploy releases, existing continuous integration and continuous delivery (CI/CD) tools can be integrated with Kubernetes.

Benefits of using Kubernetes

- A proper way of managing containers
- High availability
- Scalability
- Disaster recovery

Key Kubernetes Concepts

Node & PODs

A Kubernetes node is a machine that runs containerized workloads as part of a Kubernetes cluster. A node can be a physical machine or a virtual machine, and can be hosted on-premises or in the cloud.

A pod is composed of one or more containers that are colocated on the same host and share a network stack as well as other resources such as volumes. Pods are the foundation upon which Kubernetes applications are built.

Kubernetes uses pods to run an instance of your application. A pod represents a single instance of your application.

Pods are typically ephemeral, disposable resources. Individually scheduled pods miss some of the high availability and redundancy Kubernetes features. Instead, pods are deployed and managed by Kubernetes *Controllers*, such as the Deployment Controller.

Service & Ingress

Service is an abstraction that defines a logical set of pods and a policy for accessing them. In Kubernetes, a Service is a REST object, similar to a pod. A Service definition, like all REST objects, can be POSTed to the API server to create a new instance. A Service object's name must be a valid RFC 1035 label name.

Ingress is a Kubernetes object that allows access to the Kubernetes services from outside of the Kubernetes cluster. You configure access by writing a set of rules that specify which inbound connections are allowed to reach which services. This allows combining all routing rules into a single resource.

Ingress controllers are pods, just like any other application, so they're part of the cluster and can see and communicate with other pods. An Ingress can be configured to provide Services with externally accessible URLs, load balance traffic, terminate SSL / TLS, and provide name-based virtual hosting. An Ingress controller is in charge of fulfilling the Ingress, typically with a load balancer, but it may also configure your edge router or additional frontends to assist with the traffic.

FlowX.AI offers a predefined NGINX setup as Ingress Controller. The NGINX Ingress Controller works with the NGINX web server (as a proxy). For more information, check the below sections:

» Intro to NGINX

» Designer setup guide

ConfigMap & Secret

ConfigMap is an API object that makes it possible to store configuration for use by other objects. A ConfigMap, unlike most Kubernetes objects with a spec, has `data` and `binaryData` fields. As values, these fields accept key-value pairs. The `data` field and `binaryData` are both optional. The data field is intended to hold UTF-8 strings, whereas the `binaryData` field is intended to hold binary data as base64-encoded strings.

> ⓘ **INFO**
>
> The name of a ConfigMap must be a valid DNS subdomain name.

Secret represents an amount of sensitive data, such as a password, token, or key. Alternatively, such information could be included in a pod specification or a container image. Secrets are similar to ConfigMaps but they are designed to keep confidential data.

Volumes

A Kubernetes volume is a directory in the orchestration and scheduling platform that contains data accessible to containers in a specific pod. Volumes serve as a plug-in mechanism for connecting ephemeral containers to persistent data stores located elsewhere.

Deployment

A deployment is a collection of identical pods that are managed by the Kubernetes Deployment Controller. A deployment specifies the number of pod replicas that will be created. If pods or nodes encounter problems, the Kubernetes Scheduler ensures that additional pods are scheduled on healthy nodes.

Typically, deployments are created and managed using `kubectl create` or `kubectl apply`. Make a deployment by defining a manifest file in YAML format.

Kubernetes Architecture

Kubernetes architecture consists of the following main parts:

- Control Plane (master)
  - kube-apiserver
  - etcd
  - kube-scheduler
  - kube-controller-manager
  - cloud-controller-manager
- Node components
  - kubelet
  - kube-proxy
  - Container runtime

Install tools

kubectl

`kubectl` makes it possible to run commands against Kubernetes clusters using the `kubectl` command-line tool. `kubectl` can be used to deploy applications, inspect and manage cluster resources, and inspect logs. See the `kubectl` reference documentation for more information.

kind

`kind` command makes it possible to run Kubernetes on a local machine. As a prerequisite, Docker needs to be installed and configured. What `kind` is doing is to run local Kubernetes clusters using Docker container “nodes”.

In depth docs

» Kubernetes documentation

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to NGINX

What is NGINX?

NGINX is a free, open-source, high-performance web server with a rich feature set, simple configuration, and low resource consumption that can also function as a reverse proxy, load balancer, mail proxy, HTTP cache, and many other things.

How NGINX is working?

NGINX allows you to hide a server application's complexity from a front-end application. It uses an event-driven, asynchronous approach to create a new process for each web request, with requests handled in a single thread.

Using NGINX with FLOWX Designer

The NGINX Ingress Controller for Kubernetes - `ingress-nginx` is an ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

Ingress allows you to route requests to services based on the host or path of the request, centralizing a number of services into a single entry point.

The ingress resource simplifies the configuration of **SSL/TLS termination**, **HTTP load-balancing**, and **layer routing**.

For more information, check the following section:

» Using NGINX as a K8S ingress controller

Integrating with FLOWX Designer

FLOWX Designer is using NGINX ingress controller for the following actions:

1. For routing calls to plugins
2. For routing calls to the FLOWX Engine:

- Viewing current instances of processes running in the FLOWX engine
- Testing process definitions from the FLOWX Designer - route the API calls and SSE communications to the FLOWX engine backend
- Accessing REST API of the backend microservice

3. For configuring the Single Page Application (SPA) - FLOWX Designer SPA will use the backend service to manage the platform via REST calls

In the following section, you can find a suggested NGINX setup, the one used by FLOWX.AI:

» Designer setup guide

Installing NGINX Open Source

For more information on how to install NGINX Open Source, check the following guide:

» NGINX Install Guide

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Event-driven architecture frameworks / Intro to Redis

What is Redis?

Redis is a fast, open-source, in-memory key-value data store that is commonly used as a cache to store frequently accessed data in memory so that applications can be responsive to users. It delivers sub-millisecond response times enabling millions of requests per second for applications.

It is also be used as a Pub/Sub messaging solution, allowing messages to be passed to channels and for all subscribers to that channel to receive that message. This feature enables information to flow quickly through the platform without using up space in the database as messages are not stored.

Redis offers a primary-replica architecture in a single node primary or a clustered topology. This allows you to build highly available solutions providing consistent performance and reliability. Scaling the cluster size up or down is done very easily, this allows the cluster to adjust to any demands.

In depth docs

» Redis.io

» Redis overview

Was this page helpful?

PLATFORM OVERVIEW / Frameworks and standards / Timer expressions

When working with FLOWX.AI components, there are multiple scenarios in which timer expressions are needed.

There are two timer expressions formats supported:

- Cron Expressions - used to define the expiry date on processes
- ISO 8601 - used to define the duration of a response timeout or for a timer expression

Cron Expressions

A cron expression is a string made up of **six mandatory subexpressions (fields) that each specifies an aspect of the schedule** (for example, `* * * * * *`). These fields, separated by white space, can contain any of the allowed values with various combinations of the allowed characters for that field.

INFO

A field may be an asterisk (`*`), which always stands for “first-last”. For the day-of-the-month or day-of-the-week fields, a question mark (`?`) may be

> used instead of an asterisk.

Subexpressions:

1. Seconds
2. Minutes
3. Hours
4. Day-of-Month
5. Month
6. Day-of-Week
7. Year (optional field)

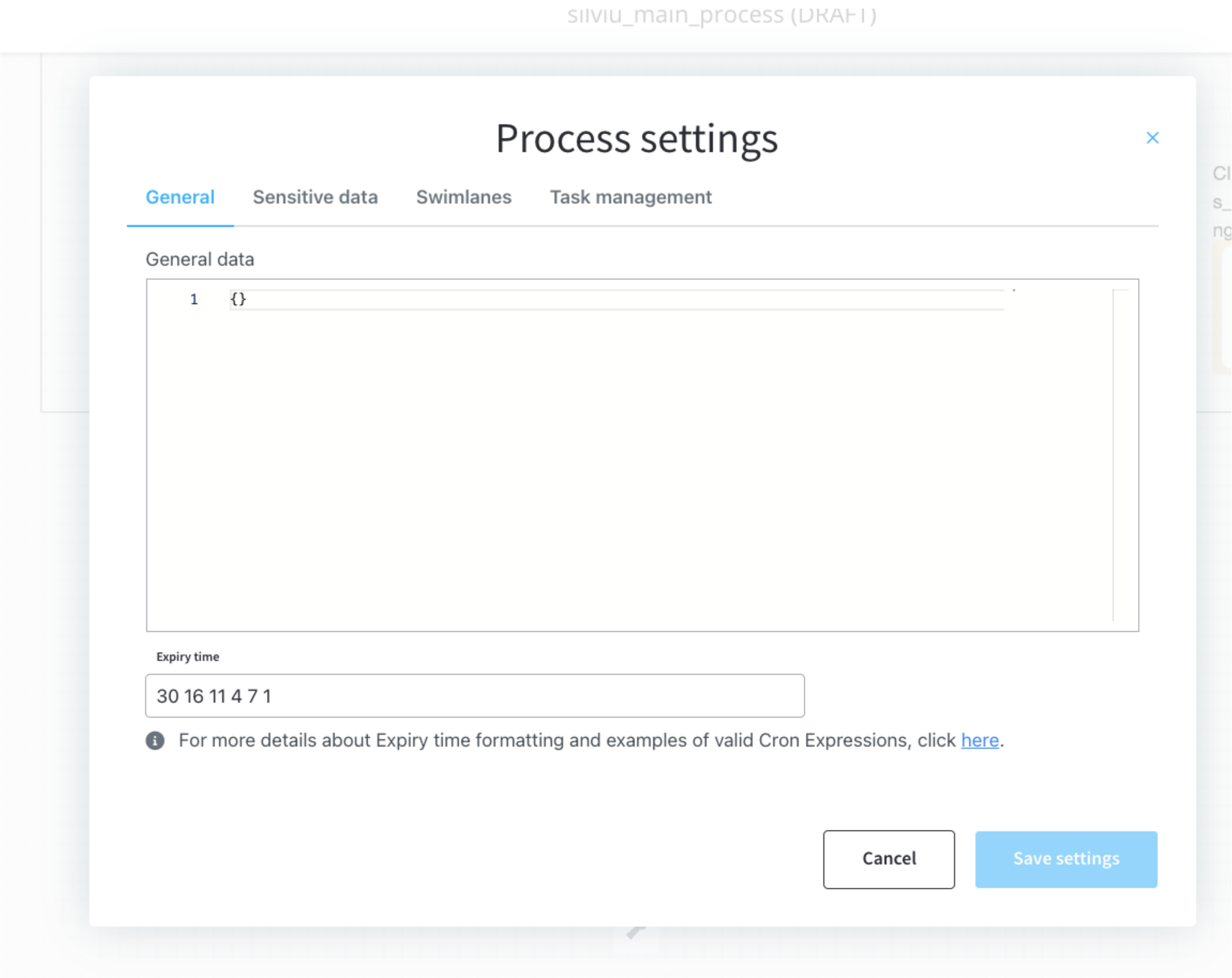
An example of a complete cron-expression is the string `0 0 12 ? * FRI` - which means **every Friday at 12:00:00 PM**.

More details:

» Scheduling cron expressions

Cron Expressions are used in the following example:

- Process definition - **Expiry time** - a user can set up a `expiryTime` function on a process, for example, a delay of 30s will be set up like:



General data

```
{}
```

Expiry time

30 16 11 4 7 1

For more details about Expiry time formatting and examples of valid Cron Expressions, click here.

ISO 8601

ISO 8601 is an international standard covering the worldwide exchange and communication of date and time-related data. It can be used to standardize the following: dates, time of delay, time intervals, recurring time intervals, etc.

More details:

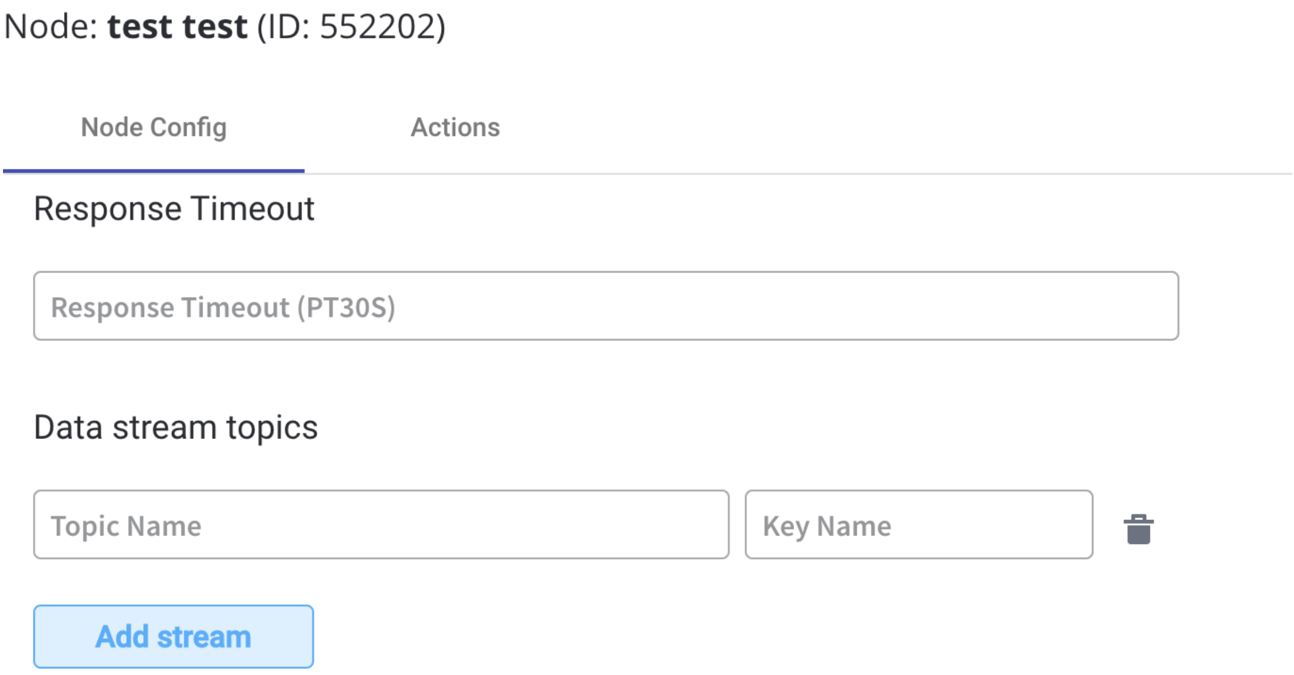
» ISO 8601

ISO 8601 format is used in the following examples:

- Node config - **Response Timeout** - can be triggered if, for example, a topic that you define and add in the **Data stream topics** tab does not respect the pattern

ISO 8601 dates and times:

Format accepted	Value ranges
Year (Y)	YYYY, four-digit, abbreviatted to two-digit
Month (M)	MM, 01 to 12
Week (W)	WW, 01 to 53
Day (D)	D, day of the week, 1 to 7
Hour (h)	hh, 00 to 23, 24:00:00 as the end time
Minute (m)	mm, 00 to 59
Second (s)	ss, 00 to 59
Decimal fraction (f)	Fractions of seconds, any degree of accuracy



Node: **test test** (ID: 552202)

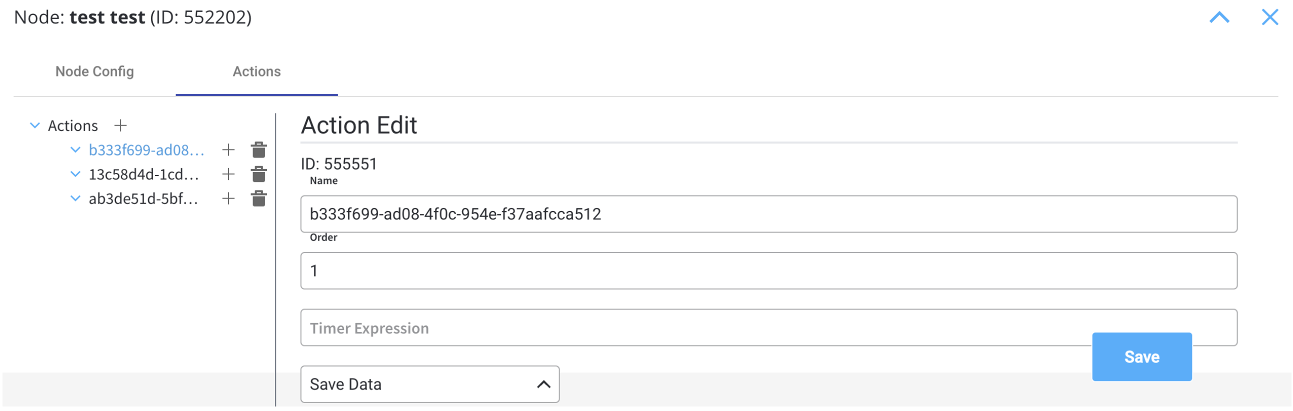
Response Timeout

Response Timeout (PT30S)

Data stream topics

Topic Name | Key Name

- **Actions** - **Timer expression** - it can be used if a delay is required on that action



Node: **test test** (ID: 552202)

Action Edit

ID: 555551

Name: b333f699-ad08-4f0c-954e-f37aafcca512

Order: 1

Timer Expression

Save Data

Was this page helpful?