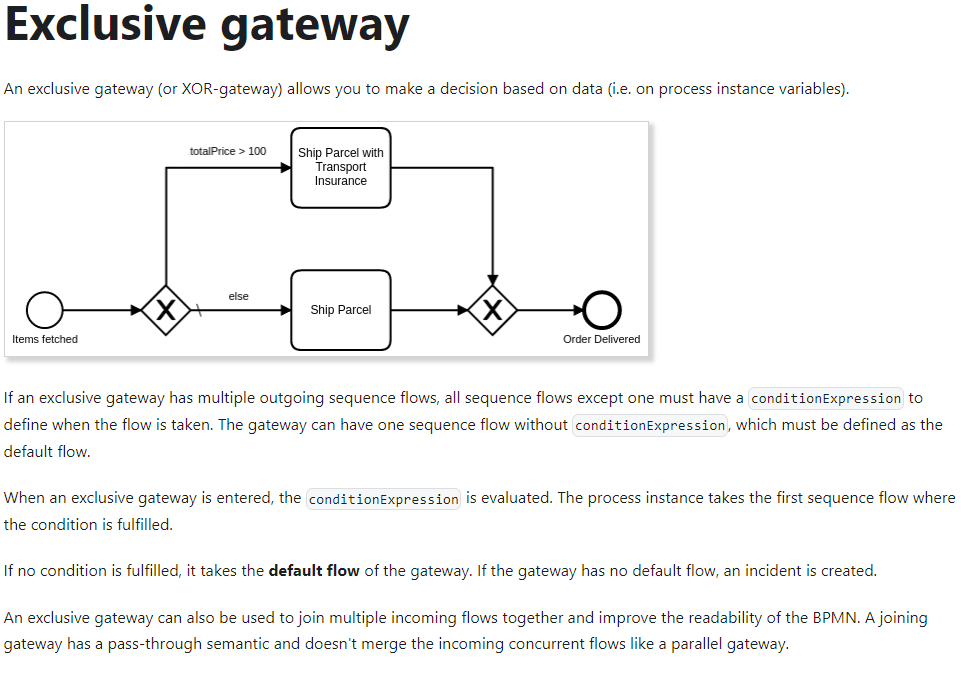
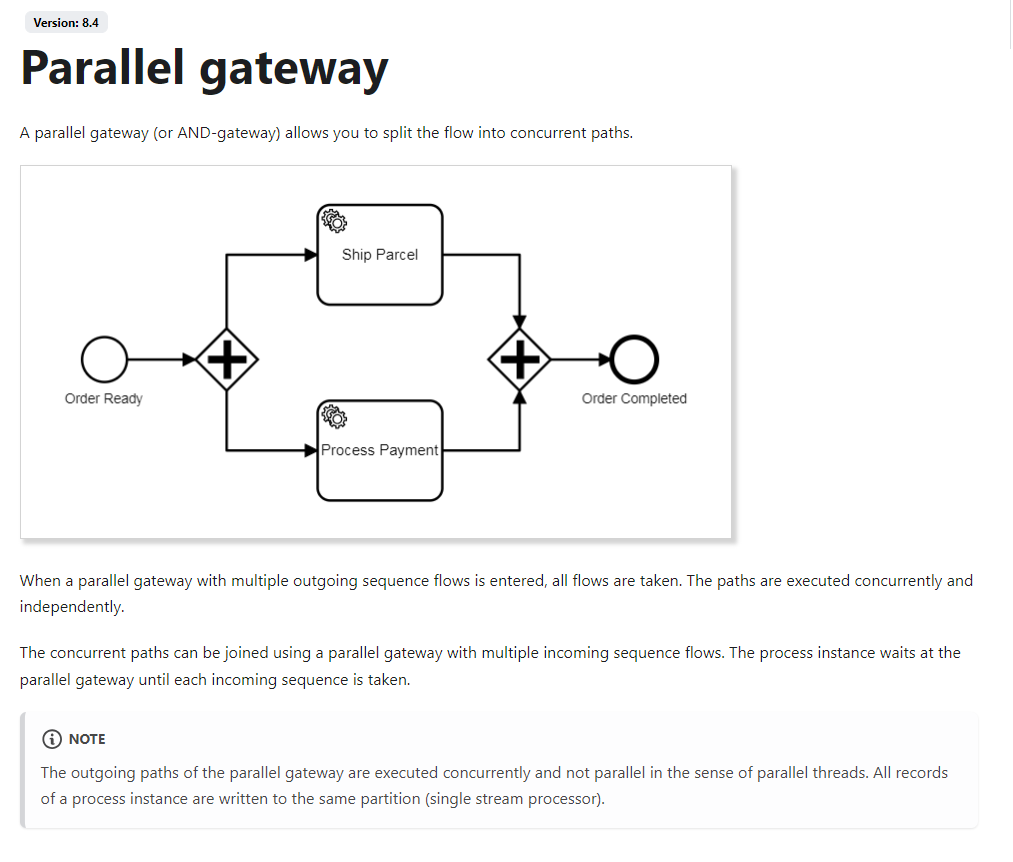
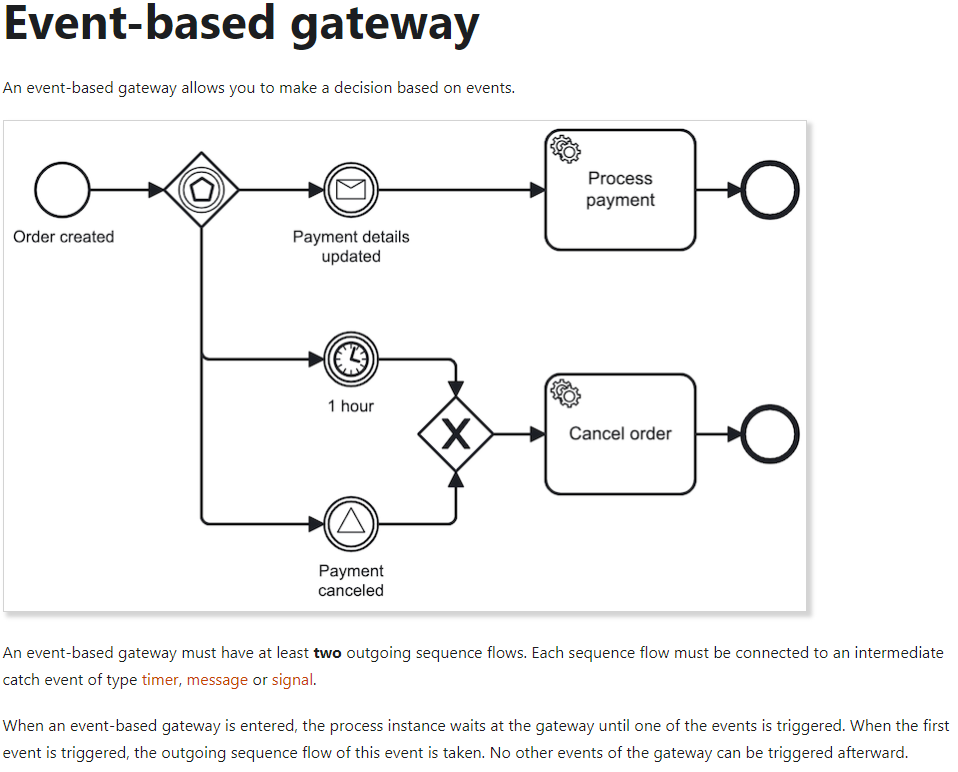


If no condition is fulfilled, it takes the **default flow** of the gateway. Note that the default flow is not expected to have a condition, and is therefore not evaluated. If no condition is fulfilled and the gateway has no default flow, an incident is created.

For example: No courses selected then the default flow is taken.







**A service task** represents a work item in the process with a specific type.

**A user task** is used to model work that needs to be done by a human actor.

**Receive tasks** reference a message; these are used to wait until a proper message is received.

**A business rule task** is used to model the evaluation of a business rule; for example, a decision modeled in [Decision Model and Notation](https://www.omg.org/dmn/) (DMN).

**A script task** is used to model the evaluation of a script; for example, a script written in Groovy, JavaScript, or Python.

**A send task** is used to model the publication of a message to an external system; for example, to a Kafka topic or a mail server.

**Service tasks**

A service task = a work item of a specific type.



When a service task is entered, a c`orresponding job is created. The process instance stops here and waits until the job is complete.

A [job worker](https://docs.camunda.io/docs/components/concepts/job-workers/) can subscribe to the job type, process the jobs, and complete them using one of the Zeebe clients. When the job is complete, the service task is completed and the process instance continues.

**Task definition**[**​**](https://docs.camunda.io/docs/components/modeler/bpmn/service-tasks/#task-definition)

A service task must have a taskDefinition. The taskDefinition is used to specify which [job workers](https://docs.camunda.io/docs/components/concepts/job-workers/) handle the service task work.

A taskDefinition specifies the following properties:

* type (required): Used as reference to specify which job workers request the respective service task job. For example, order-items.
  + type can be specified as any [static value](https://docs.camunda.io/docs/next/components/concepts/expressions/#expressions-vs-static-values) (myType) or as a FEEL [expression](https://docs.camunda.io/docs/components/concepts/expressions/) prefixed by = that evaluates to any FEEL string; for example, = "order-" + priorityGroup.
* retries (optional): Specifies the number of times the job is retried when a worker signals failure. The default is three.

The expressions are evaluated on activating the service task and must result in a string for the job type and a number for the retries.

* taskHeaders; they are static metadata handed to workers along with the job. The headers can be used as configuration parameters for the worker.

**User task**

Form

Assignment

Scheduling

**Workers**

A **job worker** is a service capable of performing a particular task in a process. Each time such a task needs to be performed, this is represented by a **job**.

A job has the following properties:

* **Type**: **defined in each task** and describes the work item. The type is referenced by workers to request the jobs they are able to perform.
* **Custom headers**: **Additional static metadata** that is defined in the process. Custom headers are used to configure reusable job workers (e.g. a notify Slack worker might read out the Slack channel from its header.)
* **Key**: Unique key to identify a job. The key is used to hand in the results of a job execution, or to report failures during job execution.
* **Variables**: The contextual/business data of the process instance required by the worker to do its work.

On requesting jobs, the following properties can be set:

* **Worker**: The identifier of the worker used for auditing purposes.
* **Timeout**: The time a job is assigned to the worker. If a job is not completed within this time, it can be reassigned by Zeebe to another worker.
* **MaxJobsToActivate**: The maximum number of jobs which should be activated by this request.
* **FetchVariables**: A list of required variable names. If the list is empty, all variables of the process instance are requested.

**After working** on an activated job, a job worker informs Camunda that the job has either completed or failed.

* When the job worker completes its work, it sends a complete job command along with any variables, which in turn is merged into the process instance. This is how the job worker exposes the results of its work.
* If the job worker can not successfully complete its work, it sends a fail job command. Fail job commands include the number of remaining retries, which is set by the job worker.
  + If remaining retries is greater than zero, the job is retried and reassigned.
  + If remaining retries is zero or negative, an incident is raised and the job is not retried until the incident is resolved.

**Connectors**

**Connectors vs workers**

* Connector itself is environment-agnostic. There is a runtime environment for Camunda 8 SaaS that can wrap and call this Connector.
* A Connector only consists of core business functionality. No environment tasks, no Camunda 8 job worker-related code. The Connector only needs input variables and access to secrets so they can be used in defined input attributes.  
  Workers are complete Zeebe applications
* Context

**Which one should you choose?**

* Need access to a low-level API in Camunda 8 to perform a very specific task? You are better off with job workers.
* Want to write your worker logic in something other than Java? Job workers are your way to move forward.
* Want to create worker logic that is reusable in any environment? Write a Connector.
* Want to focus on your worker's logic and have no need for using low-level Camunda 8 API? Write a Connector.
* Want to provide a standardized modeling experience alongside your runtime behavior? Write a Connector.

<https://docs.camunda.io/docs/components/concepts/outbound-connectors-job-workers/#:~:text=Instead%2C%20a%20Connector%20is%20delivered,executed%20to%20work%20on%20jobs>.

**Custom connectors**[**​**](https://docs.camunda.io/docs/components/modeler/bpmn/service-tasks/#task-definition)

Inbound / outbound

Created as separate projects

Template + logic

Camunda

h2 db engine - поэтому быстрая, есть многопоточка, легкая (2.5 мб)

Zeebe

rockdb Под капотом (распределенная). Повышает мощность за счет увеличения инстансов

