Advanced Software Engineering (LAB)

Stefano Forti

name.surname@di.unipi.it

Department of Computer Science @ University of Pisa


Questions?

Déjà vu

Coreography vs. Orchestration

Coreography

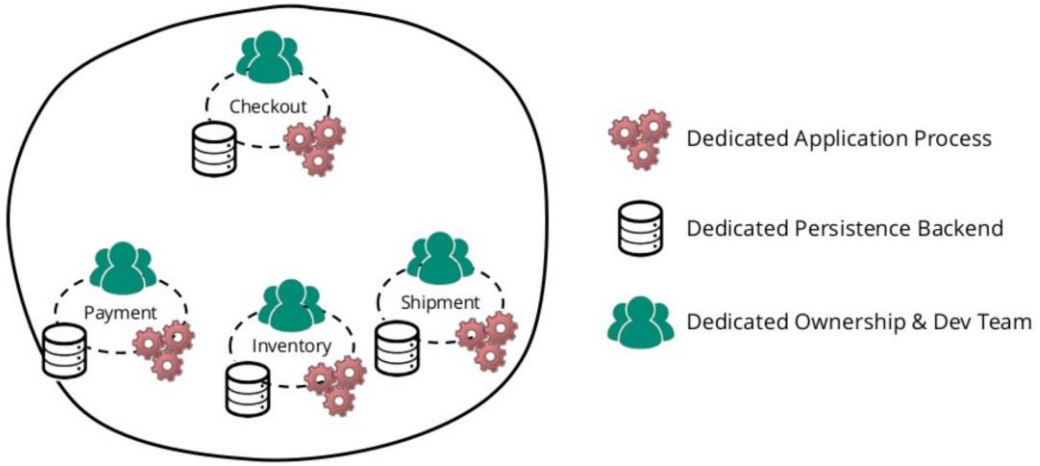
- Service choreography permits to services to self-coordinate in a P2P fashion.

Orchestration

- Orchestration is the centralised and automated coordination of services.

Microservice Development

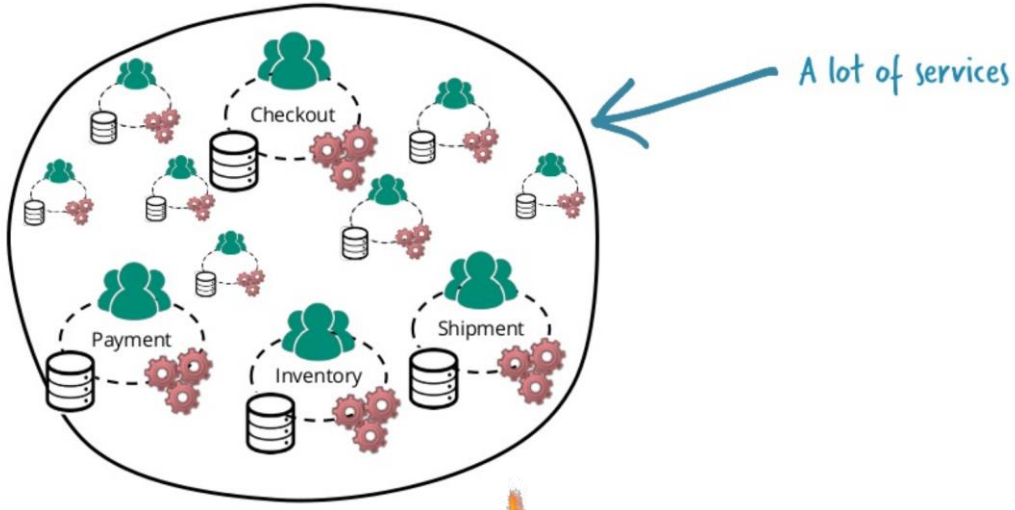
- We can design and implement a set of dedicated, autonomous tasks to do business tasks in our company domain.



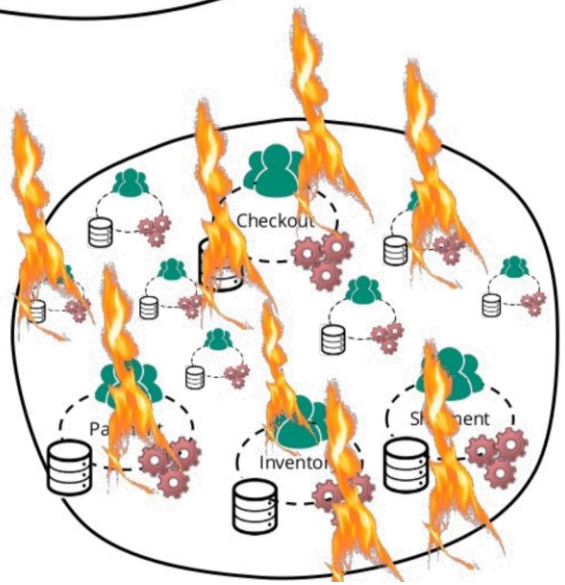
- Each will have its own business logic, data, and a (careful) team maintaining it.

Where have you experienced complexity?

- Complexity, in modern software systems, lies in the collaboration among services (especially when they are a lot!).

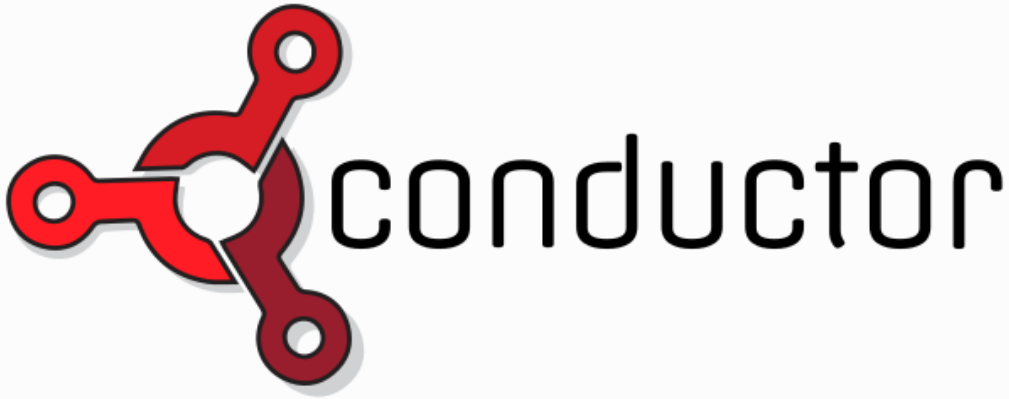


- Everything might easily “fail fast” when micro-services do not manage to properly coordinate.



Netflix orchestrates…

https://netflix.github.io/conductor/



NETFLIX | OSS

Conductor is an *orchestration* engine that runs in the cloud.

Why not peer to peer choreography?

With peer to peer task choreography, we found it was harder to scale with growing business needs and complexities. Pub/sub model worked for simplest of the flows, but quickly highlighted some of the issues associated with the approach:

- Process flows are "embedded" within the code of multiple application.
- Often, there is tight coupling and assumptions around input/output, SLAs etc, making it harder to adapt to changing needs.
- Almost no way to systematically answer "how much are we done with process X"?

Not everyone is Google

Life *beyond* Distributed Transactions

AN APOSTATE'S OPINION

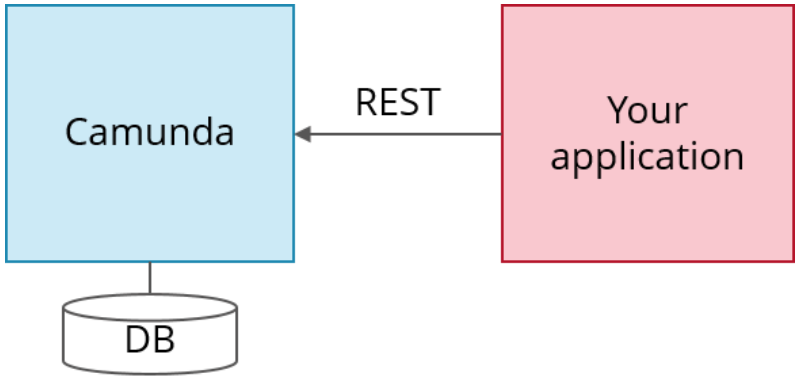
PAT HELLAND

This is an updated and abbreviated version of a paper by the same name first published in CIDR (Conference on Innovative Database Research) 2007.

Many applications are implicitly being designed with both entities and activities today. They are simply not formalized, nor are they used consistently. Where the use is inconsistent, bugs are found and eventually patched. By discussing and consistently using these patterns, better large-scale applications can be built and, as an industry, we can get closer to building solutions that allow business-logic programmers to concentrate on the business problems rather than the problems of scale.

camunda

- Camunda is a framework supporting BPMN for workflow and process automation.
- It provides a RESTful API which allows you to use your language of choice.



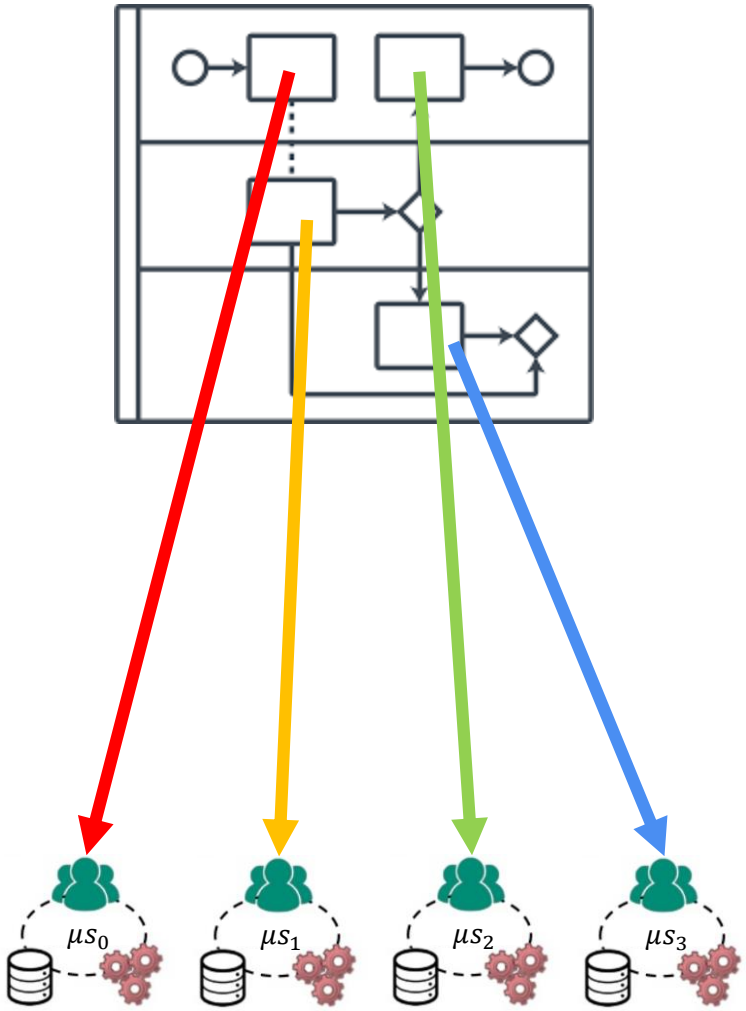
- Workflows are defined in BPMN which can be graphically modeled using the Camunda Modeler.

Who uses Camunda (only some…)?

How does it work? (A)

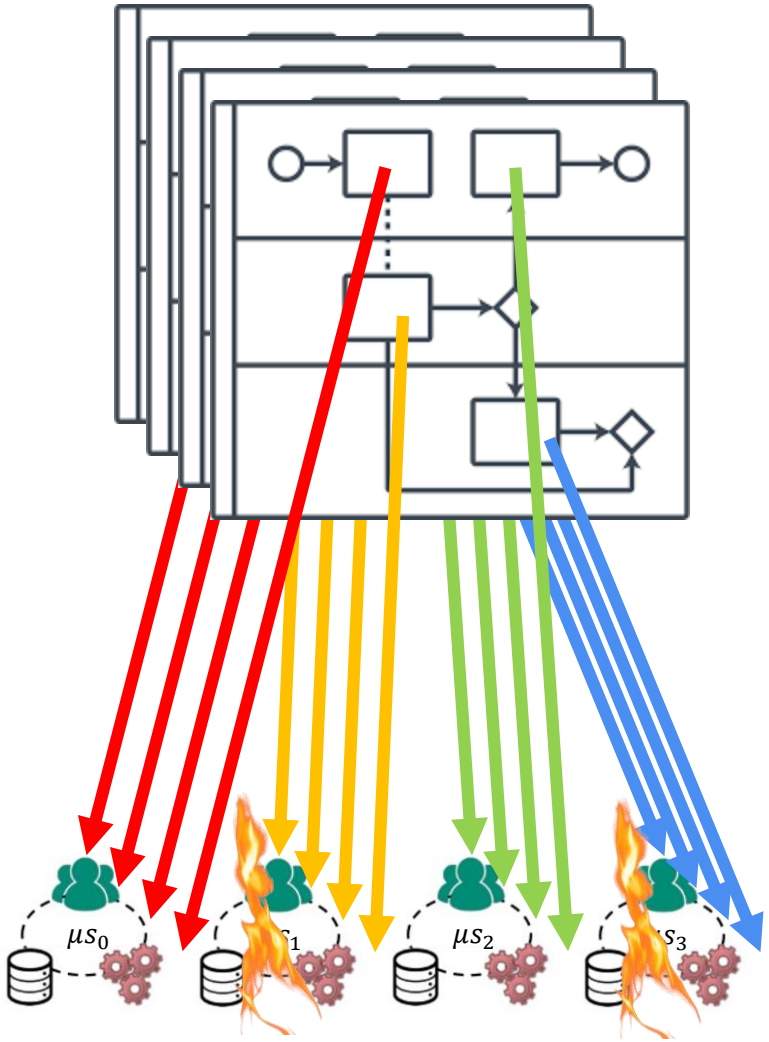
After defining a BPMN process, Camunda can directly call services via built-in *connectors*.

It supports both RESTful and SOAP services in this way.



Scaling (A)

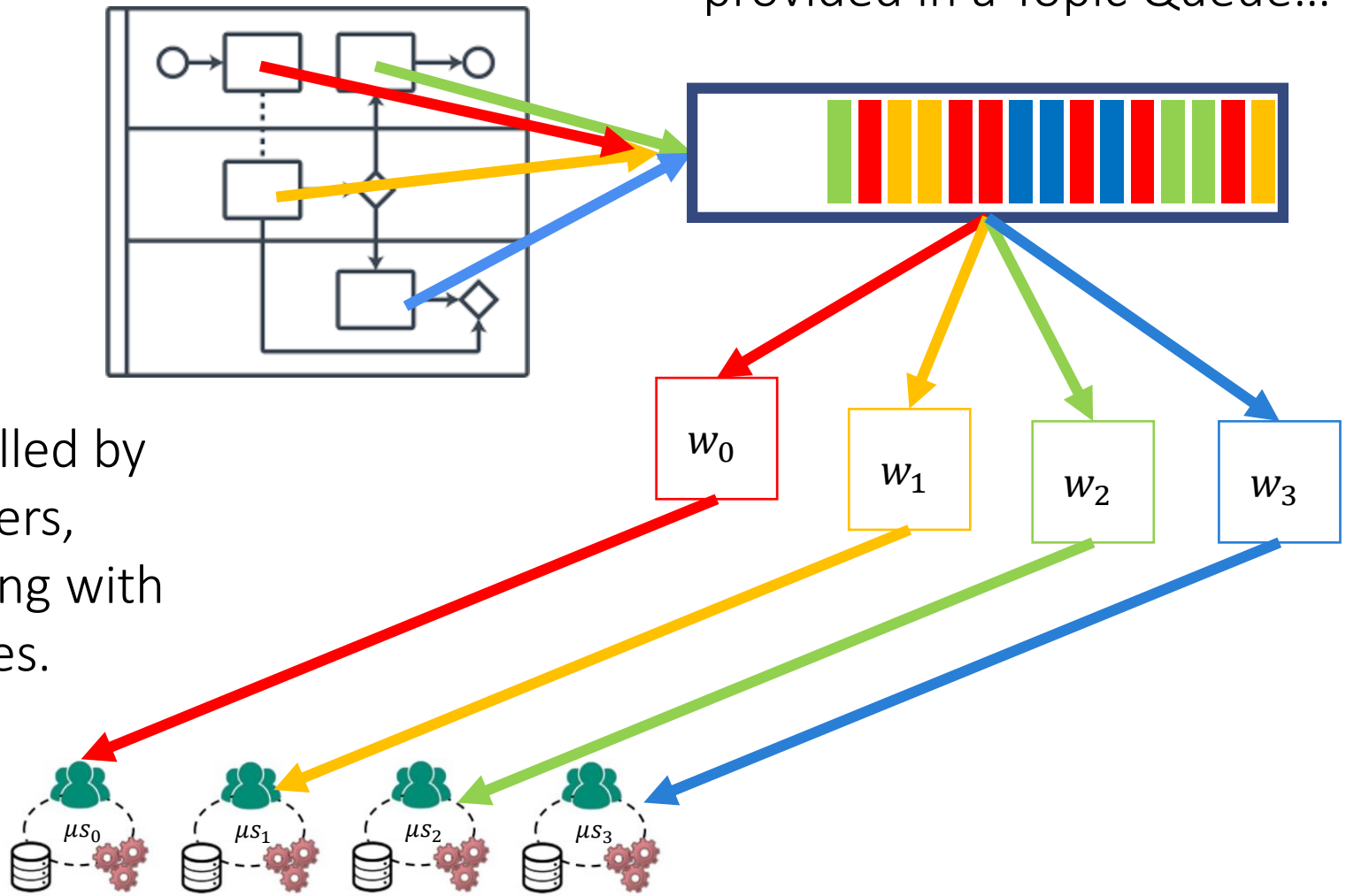
However, it only allows scaling on process instances, NOT on microservices.



How does it work? (B)

A more interesting pattern is known as *External Task*.

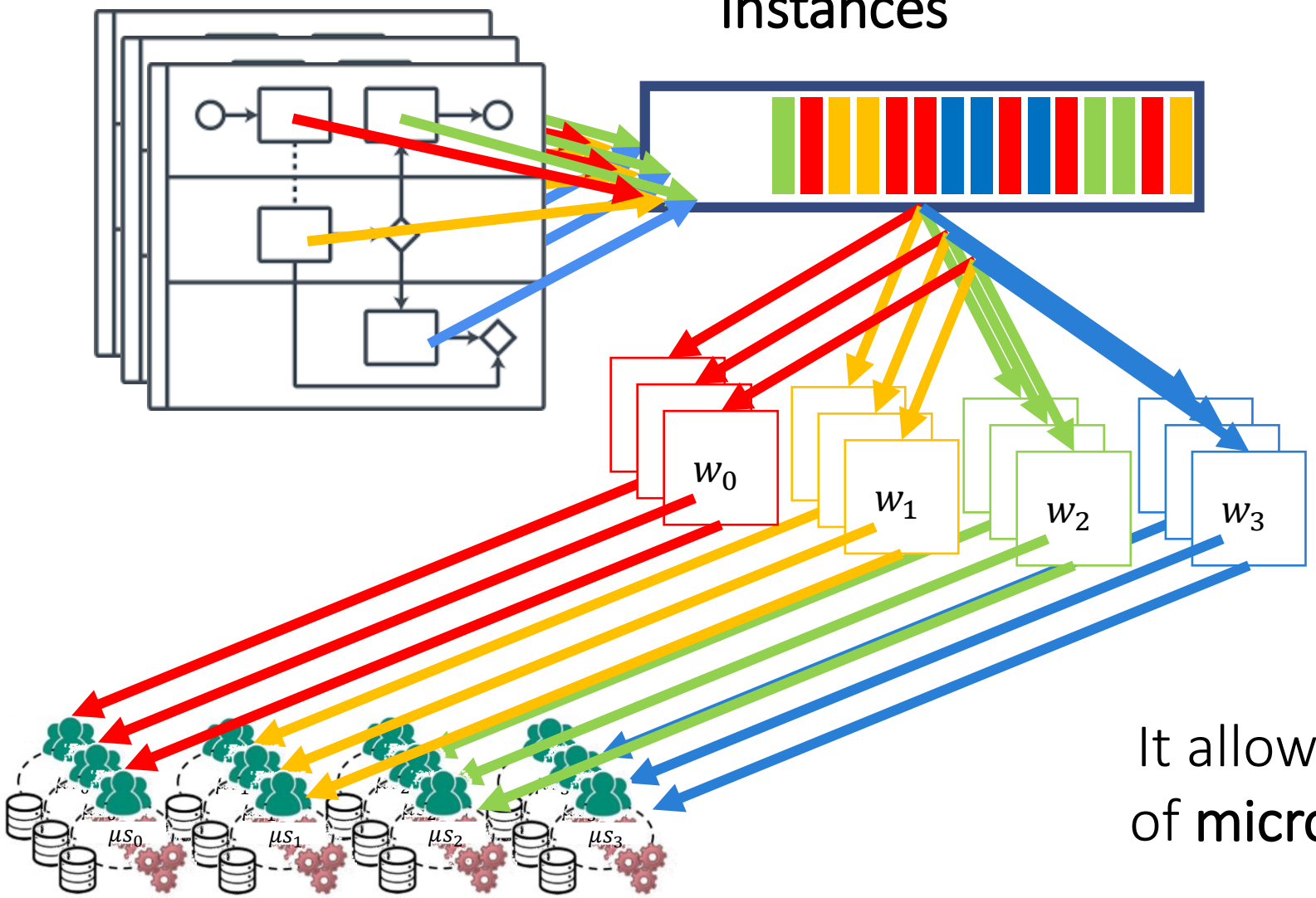
Units of work (Tasks) are provided in a Topic Queue…



…that can be polled by RESTful workers, possibly interacting with microservices.

Scaling (B)

It allows scaling of **process instances**



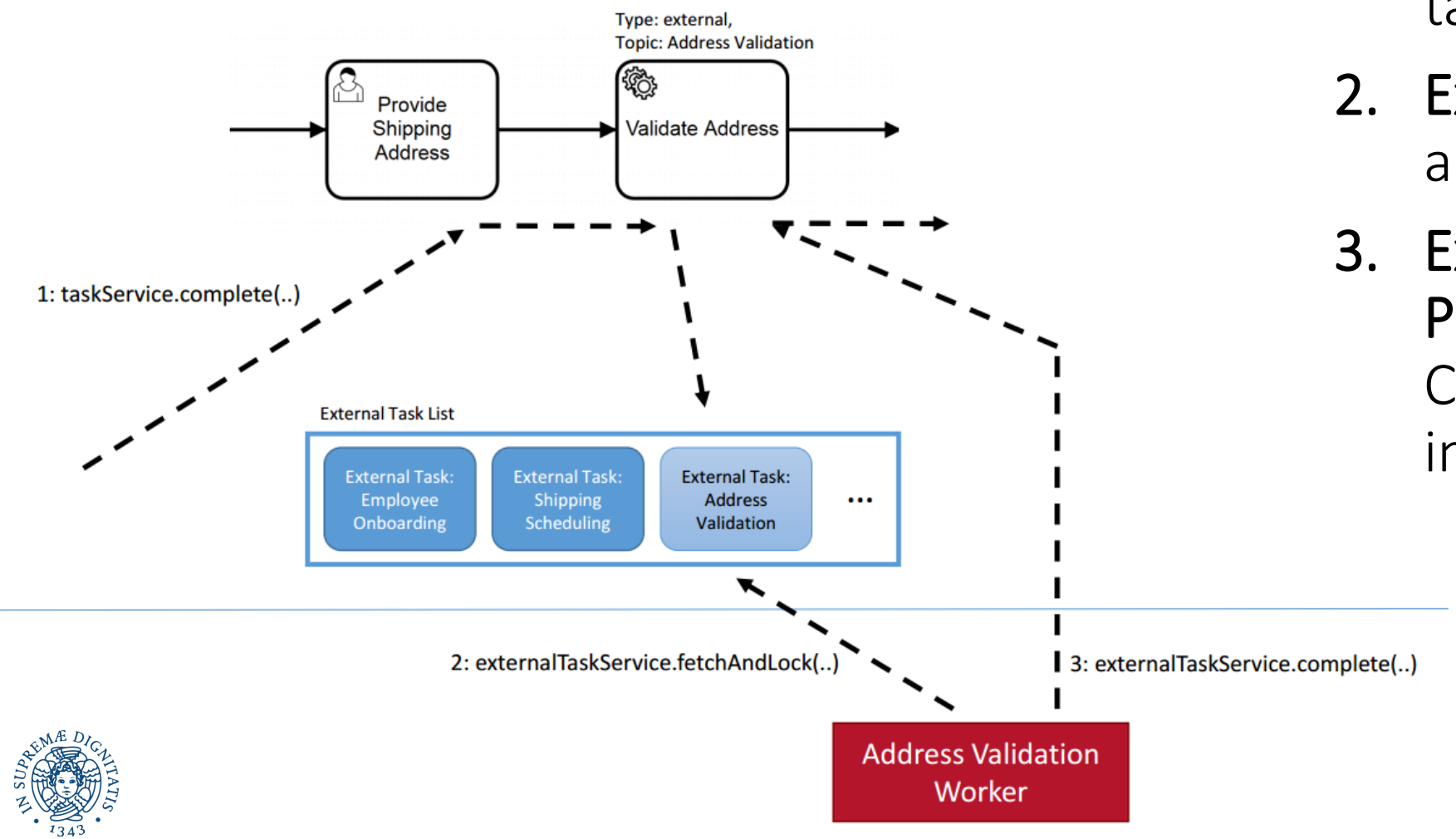
It allows scaling of **workers**

It allows scaling of **microservices**

The Workers Philosophy

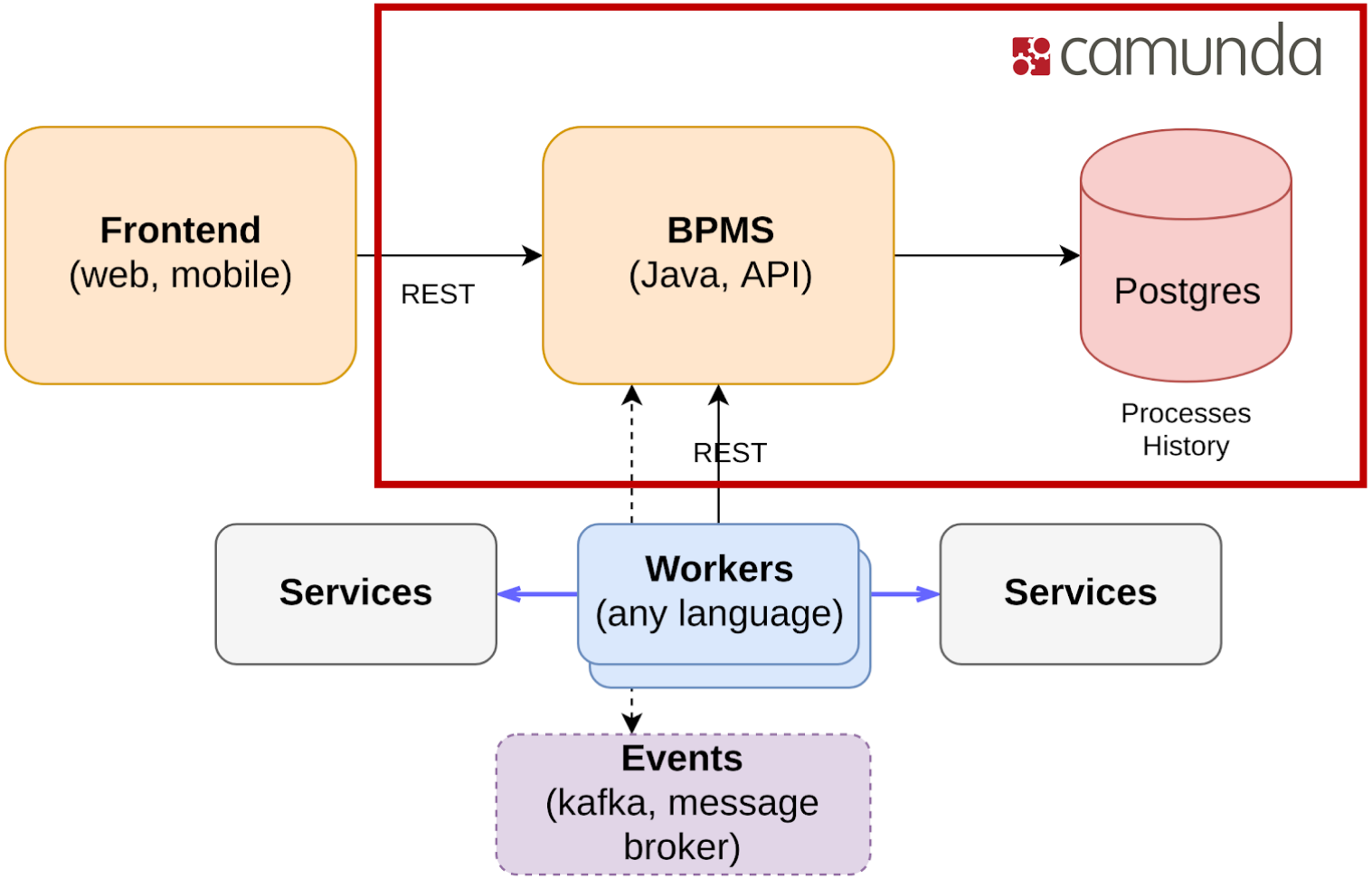
You can think of them as “users” involved in the business process.

Step-by-step



1. **Process Engine**: Creation of an external task instance
2. **External Worker**: Fetch and lock external tasks
3. **External Worker & Process Engine**: Complete external task instance

Software with Camunda

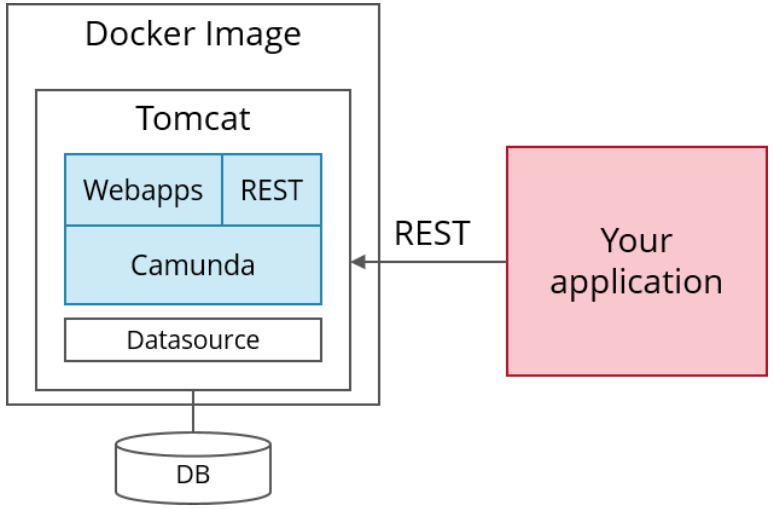


camunda in Docker

- We can use Docker to run **Camunda BPM Platform**:

```
docker pull camunda/camunda-bpm-platform:latest
docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
```

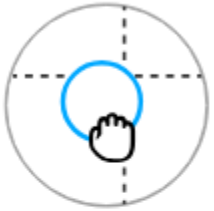
- Browse `127.0.0.1:8080/camunda` and enter credentials `demo demo`.



Camunda Modeler

- Install the Camunda Modeler from https://camunda.com/download/modeler/

- And now *Model-Execute-Enjoy*



Model

Create BPMN workflow diagrams and DMN decision tables in an editor that both business users and developers love to use.

Execute

Execute your workflows and decisions in powerful engines that are paired with essential applications for process automation projects.

Enjoy

Never fear Business Process Management again as you will love Camunda. If you find that hard to believe, you should just give it a try.

Agenda

- Build a model with the Modeler
- Deploy it to the Engine
- Start it with Tasklist
- Manipulate the process with Cockpit

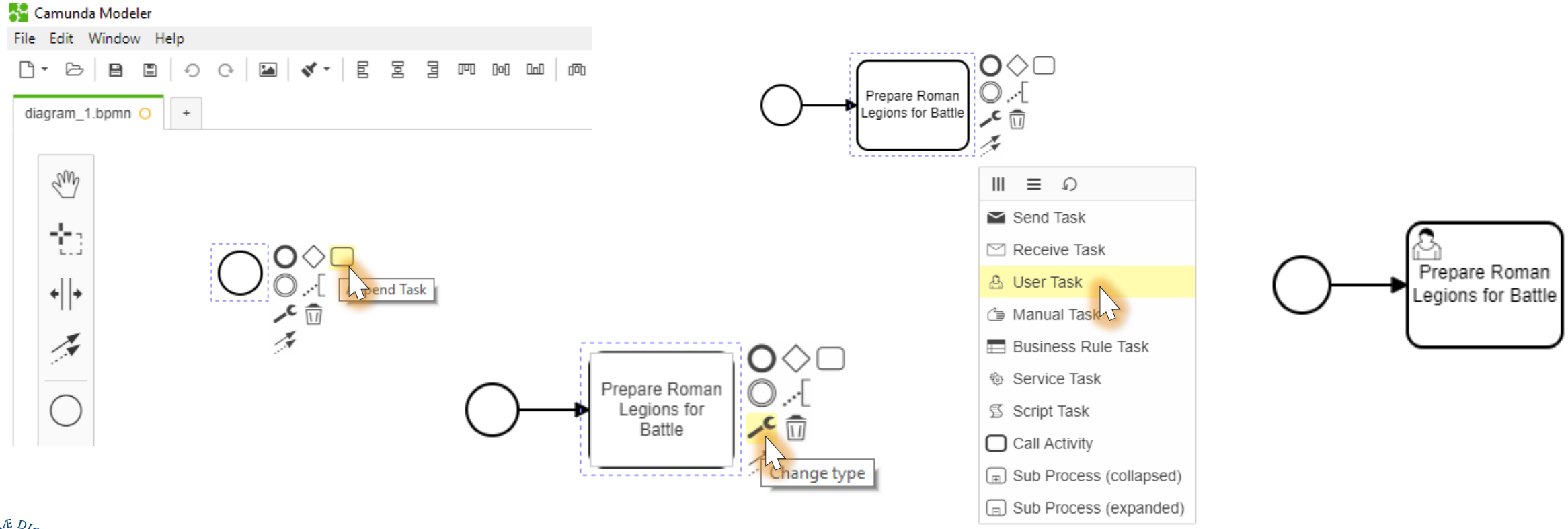
Start the Modeler

- First 

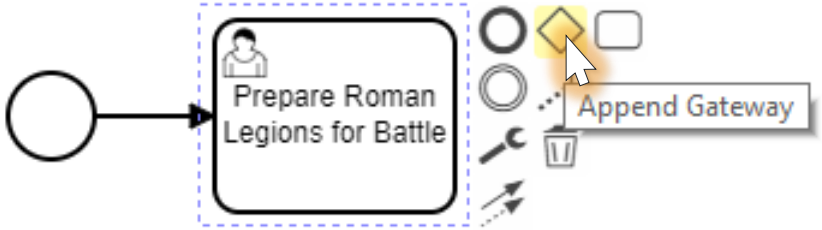
- As a modelling best practice, always draw your diagrams left to right →
- Some people can't get used to going from the top to the bottom…

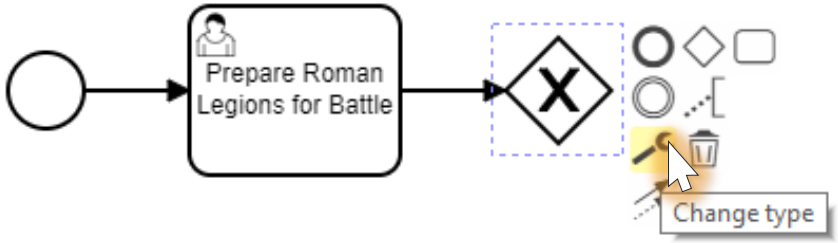
Create a User Task

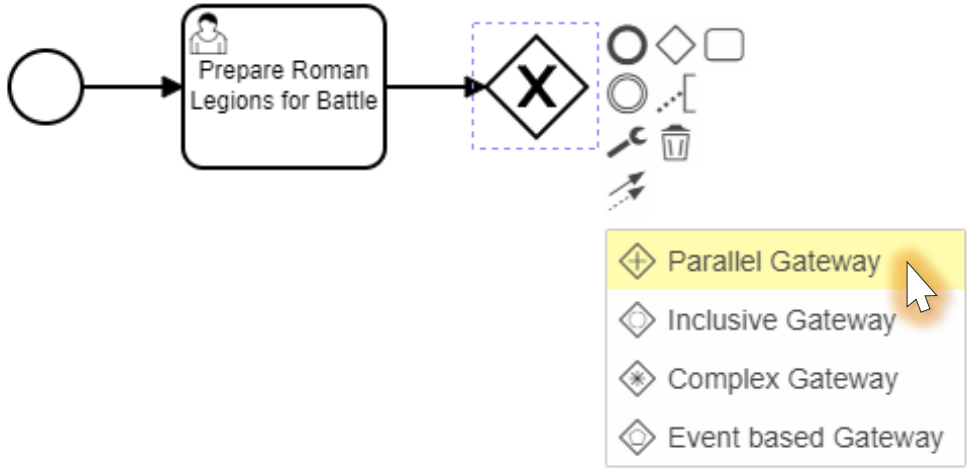
- To create a User Task (i.e., involving humans), follow these steps:



Add a parallel gateway

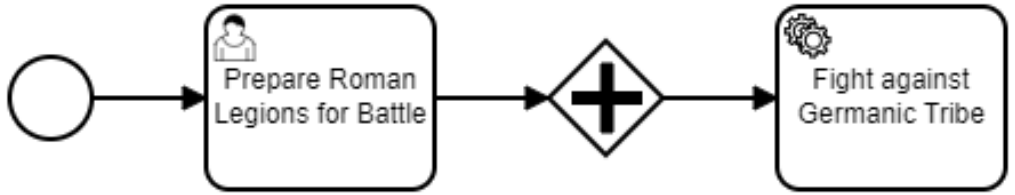




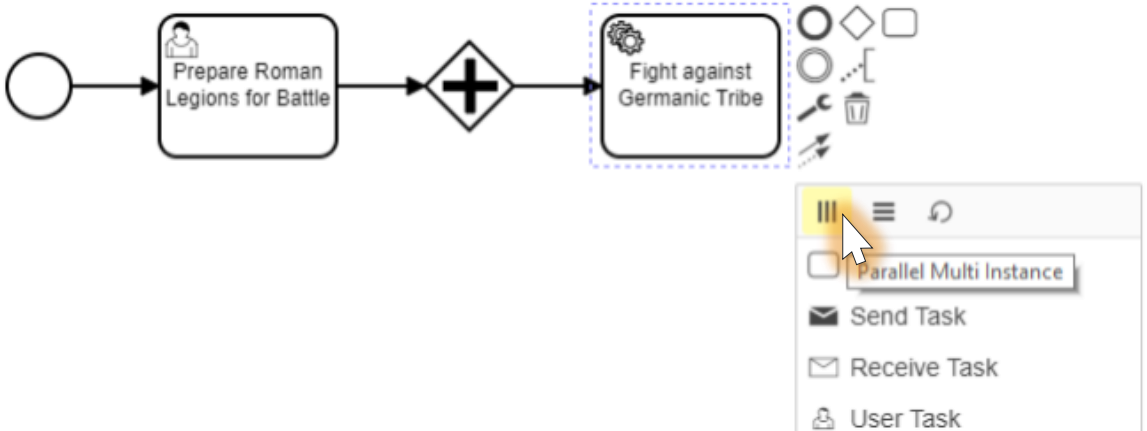


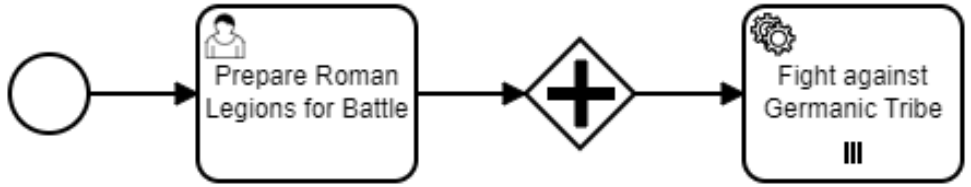
Add a Service Task

- Simply select the **Service Task** type



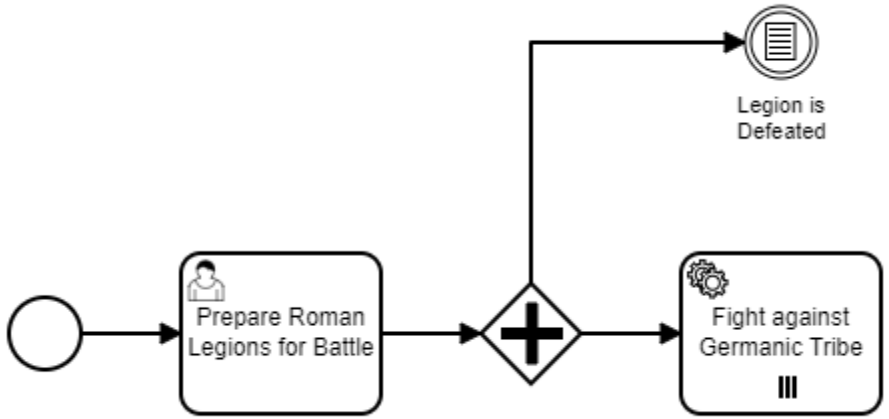
- Then, make it **Parallel Multi-Instance** (we can have many Germanic Tribes attacking our Legion).





Conditional Events

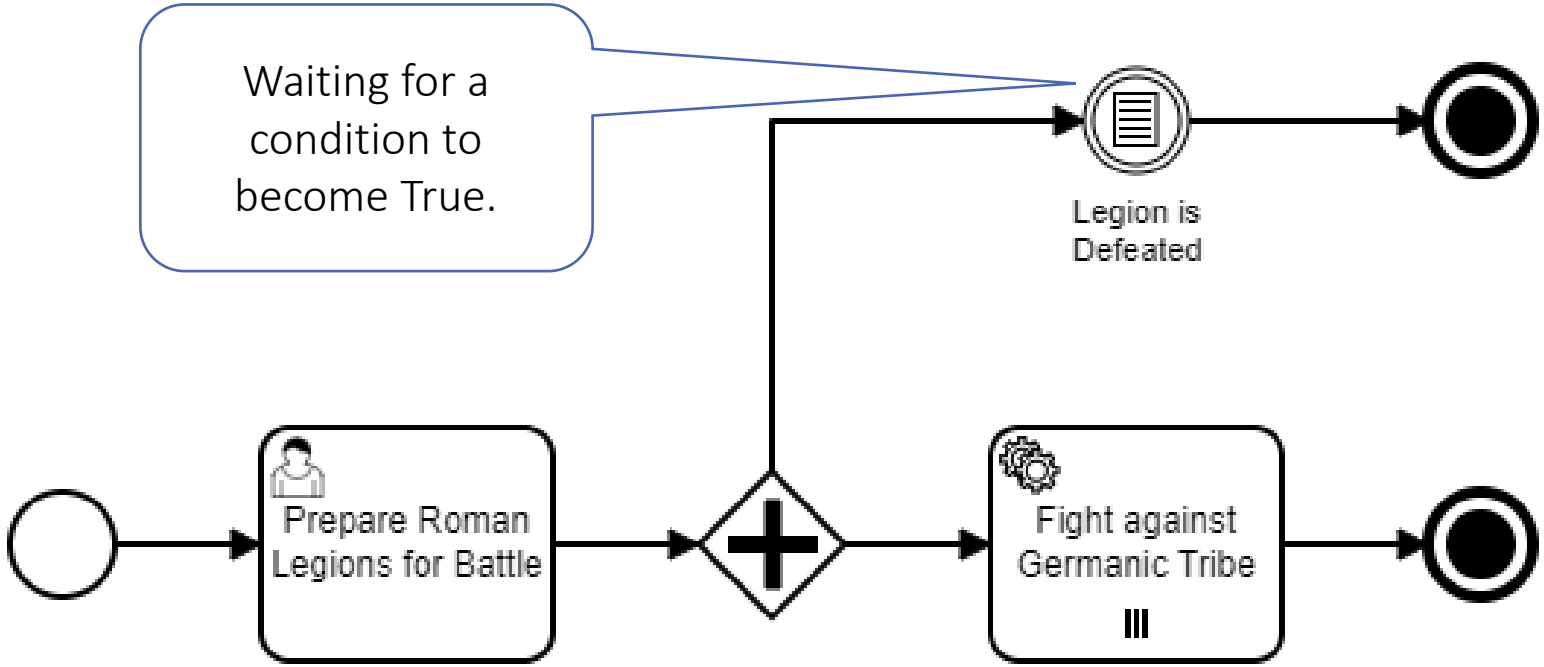
- Add an **Intermediate Event** and make it **Conditional** (what if we lose the battle?)



- Append an **End Event** and make it a **Terminate Event**.

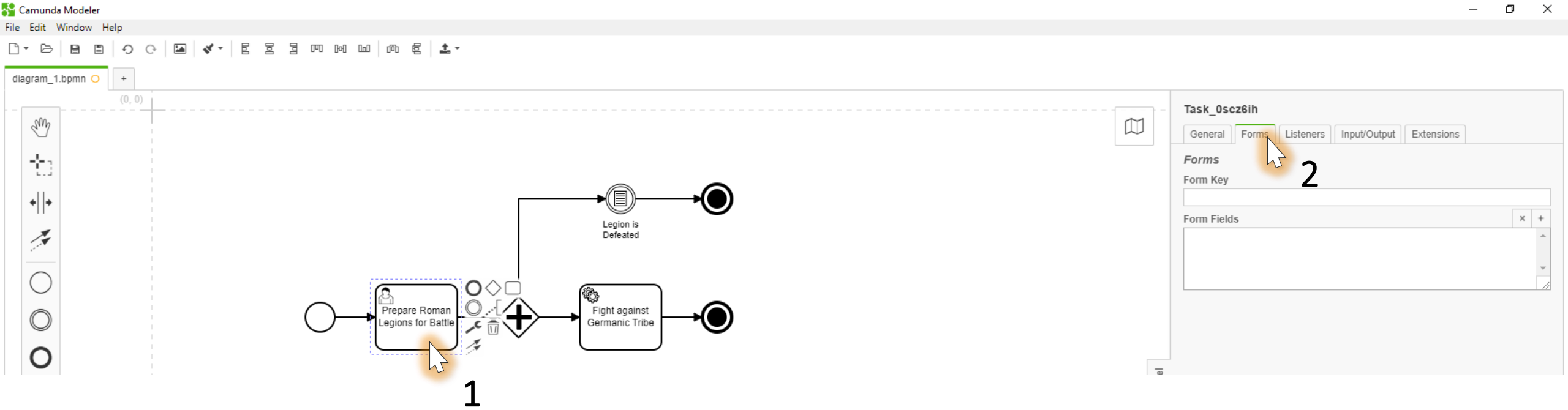
The battle terminates if we win too…

- The whole process should stop in both cases (defeated or won).



Configuring User Tasks

- Forms permit to quickly prototype a UI for our process.

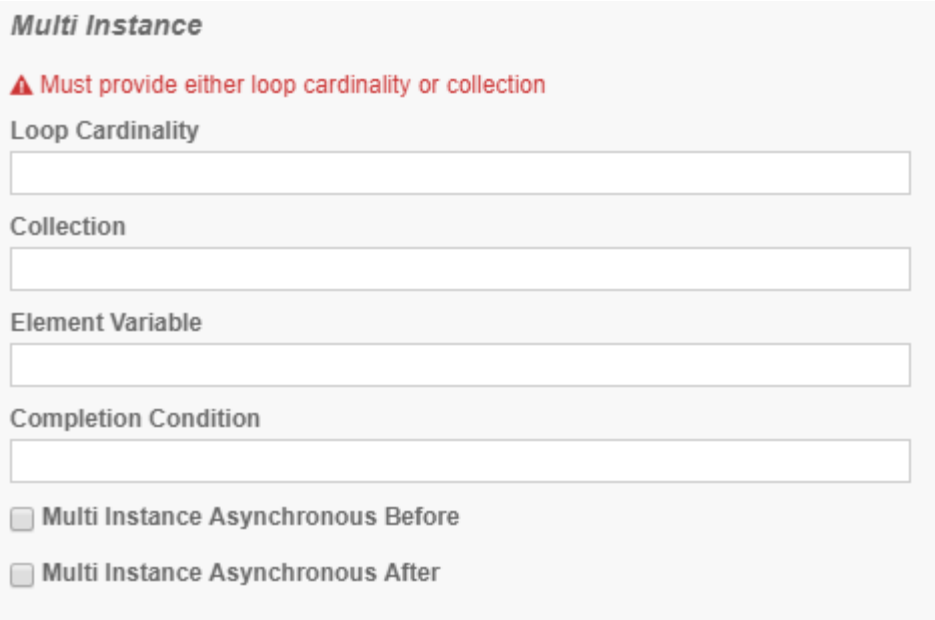


Forms

- Click on + and add a variable with
  - ID - `numberOfTribes`
  - Type - `long`
  - Label - `How many tribes are there?`
  - Default Value - `10`

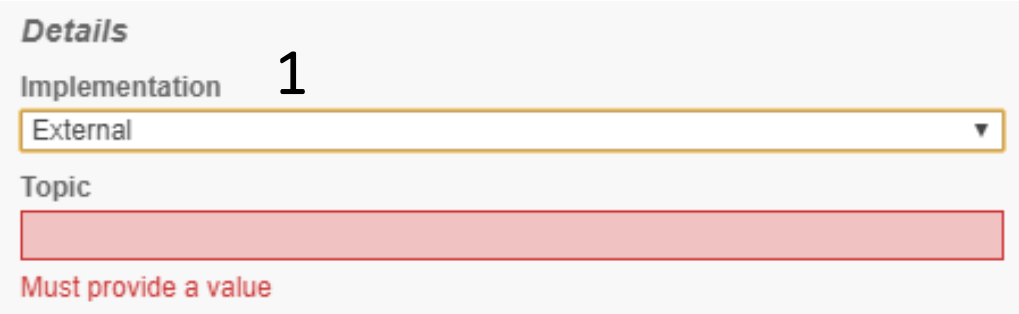
Multi-instance

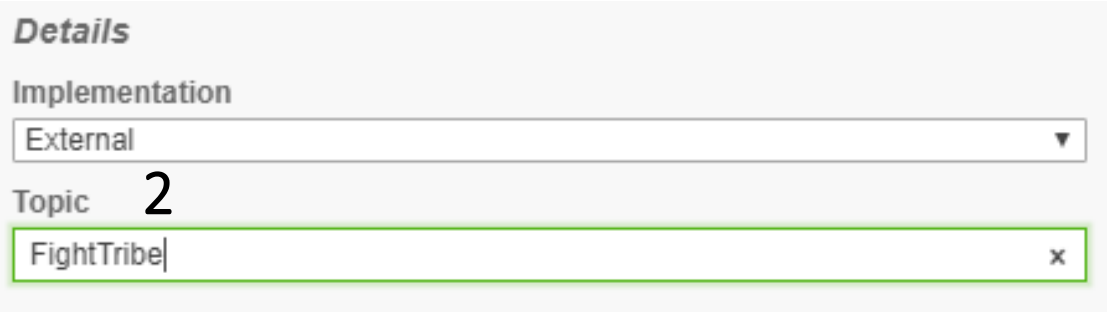
- For the multi-instance we should set the **Loop Cardinality** to `numberOfTribes`
- We identify the value of a variable as `#{numberOfTribes}`.



Task

- Now, we should choose an **Implementation** for our class.
- We will (avoid Java and) exploit an **External** worker.

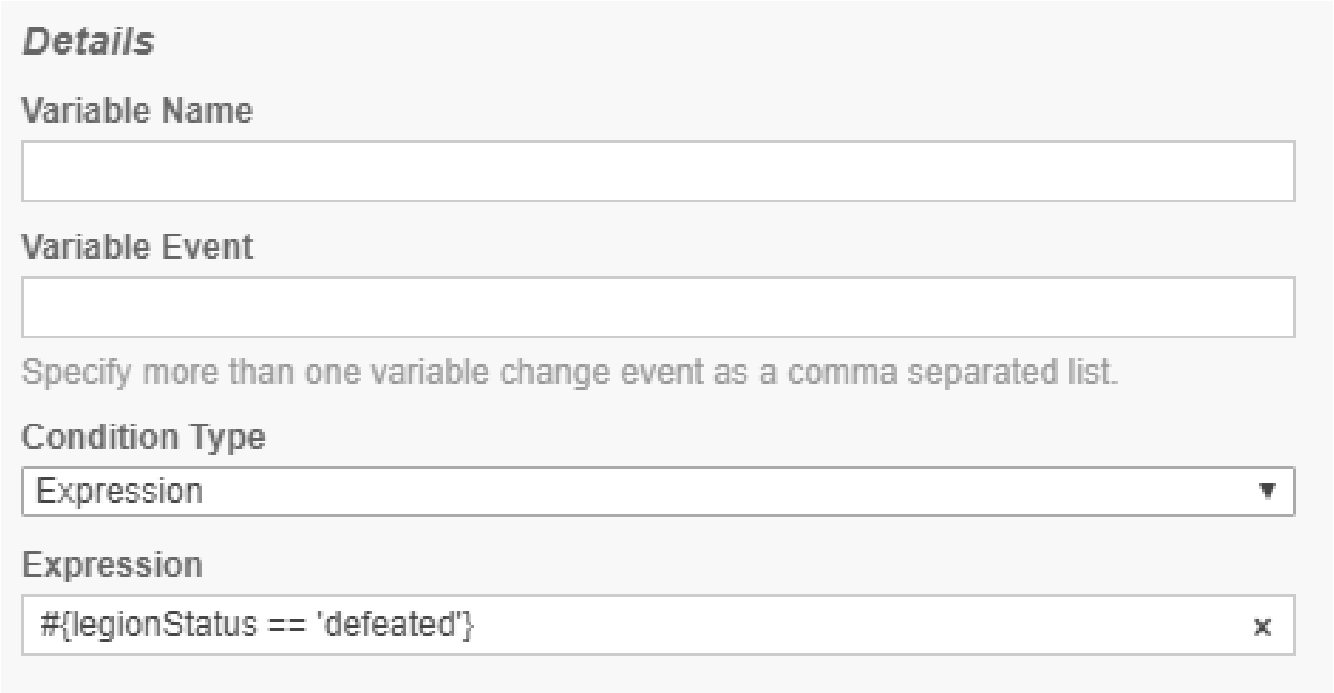




- The External worker (in JS) will subscribe to the topic `FightTribe` and get tasks from there.

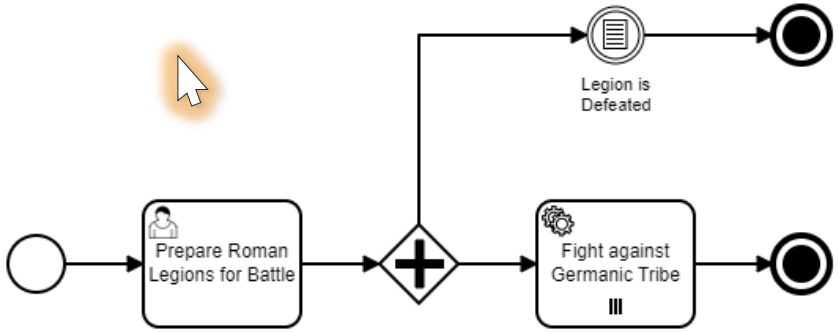
Conditions

- This condition verifies when the legion is defeated, i.e. when the *Expression* `#{legionStatus == 'defeated'}` evaluates to True.

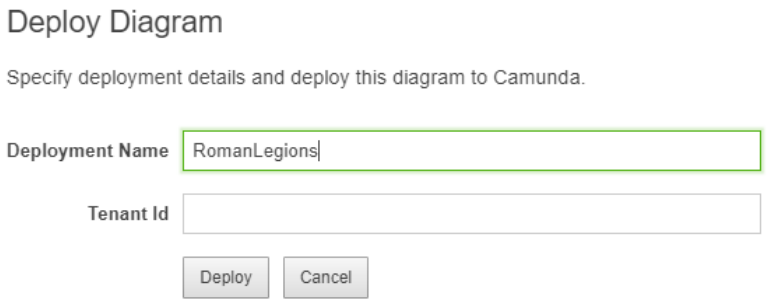


Deploy

- Give the whole process a name by clicking in any empty part of the window and completing the Properties of the process as shown here:

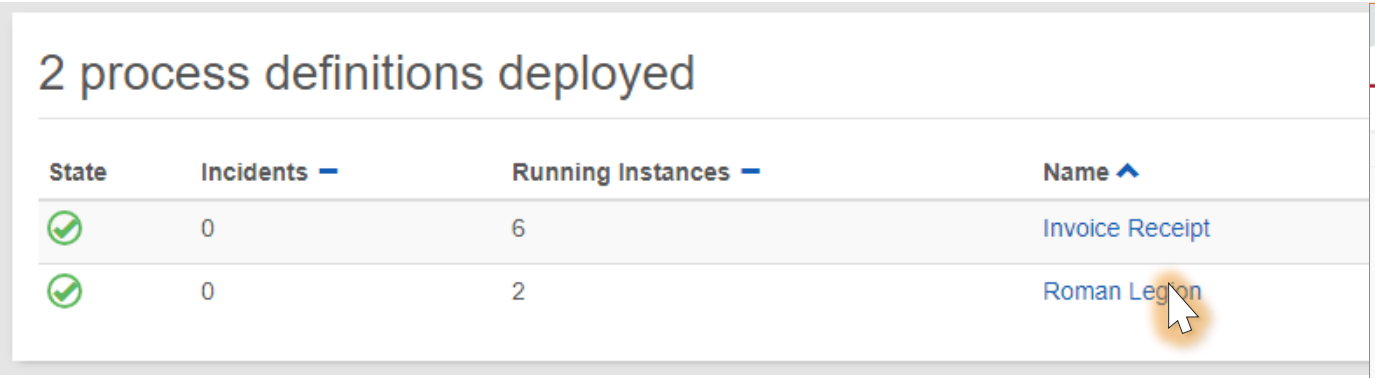


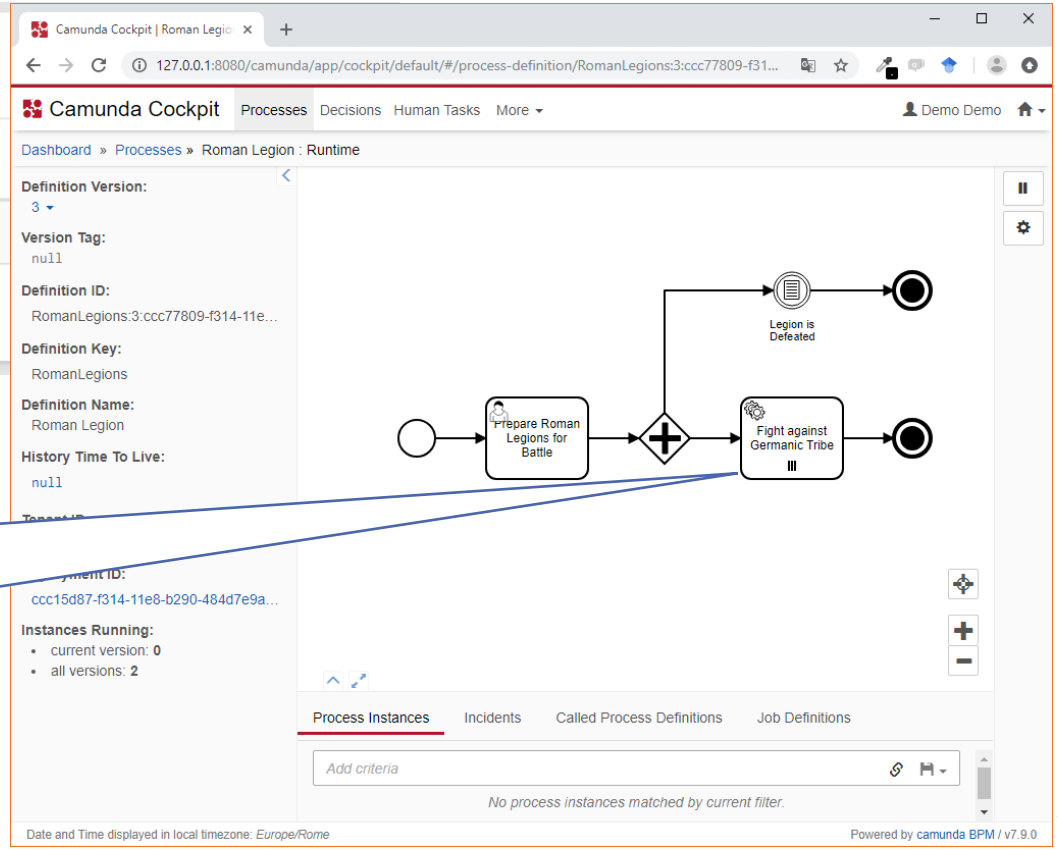
- **Save** and click on **Deploy Current Diagram**



Deployed Processes

- Enter in Camunda Cockpit and click on **Processes** at the top of the screen.





We are left with creating a **Worker** for this task to complete.

External Tasks

- They exploit the REST API of Camunda [https://docs.camunda.org/manual/7.9/reference/rest/external-task/]
- Particularly:

Fetch and Lock

POST `/external-task/fetchAndLock`

Ask for a task to compute (and lock it to prevent other worker from doing the job)

Complete

POST `/external-task/{id}/complete`

Tell that a task has been completed to Camunda ☺

Fetch and Lock

https://docs.camunda.org/manual/7.9/reference/rest/external-task/fetch/

```
{
  "workerId":"aWorkerId",
  "maxTasks":2,
  "usePriority":true,
  "topics":
      [{"topicName": "createOrder",
      "lockDuration": 10000,
      "variables": ["orderId"]
      }]
}
```

Complete

https://docs.camunda.org/manual/7.9/reference/rest/external-task/post-complete/

```
{
  "workerId": "aWorker",
  "variables":
      {"aVariable": {"value": "aStringValue"},
      "anotherVariable": {"value": 42},
      "aThirdVariable": {"value": true}},
  "localVariables":
      {"aLocalVariable": {"value": "aStringValue"}}
}
```

Workers in Javascript

- To create a NodeJS project

```
mkdir romanlegions
cd ./romanlegions
npm init romanlegions -y
```

- Install the Camunda **External Task Client JS library**:

```
npm install -s camunda-external-task-client-js
```

- Docs are on GitHub [https://github.com/camunda/camunda-external-task-client-js]

Worker Template

```
const { Client, Variables, logger } = require('camunda-external-task-client-js');

// configuration for the Client:
//  - 'baseUrl': url to the Process Engine
//  - 'logger': utility to automatically log important events
const config = { baseUrl: 'http://localhost:8080/engine-rest', use: logger, maxTasks:1};

// create a Client instance with custom configuration
const client = new Client(config);

// susbscribe to the topic: 'FightTribe'
client.subscribe('FightTribe', async function(params) {
  task = params['task']
  callback = params['taskService']

  // Business Logic
  const processVariables = new Variables();


  // Callback - Complete
  await callback.complete(task, processVariables, null);

});
```

Our business logic

```
// Business Logic
const processVariables = new Variables();

if (Math.random() > 0.9){
  console.log('[Germanic Tribe Fighter] The battle has been lost!');
  processVariables.set("legionStatus", "defeated");
} else {
  console.log('[Germanic Tribe Fighter] The Roman Legion won the battle!');
  processVariables.set("legionStatus", "victorious")
}

// Callback - Complete
await callback.complete(task, processVariables, null);
```

```
node worker.js
```

See what happens in the Cockpit