EPI Framework Tutorial

Tim Müller (CCI Group, UvA)

t.muller@uva.nl

ICT.OPEN - 20 April 2023

Welcome!

- 12:30 - 12:45: **Introduction** *(presentation)*
- 12:45 - 13:30: **Part 1: Hello, world!** *(guided hands-on)*
- 13:30 - 13:45: **Break**
- 13:45 - 14:15: **Part 2: A workflow for Disaster Tweets** *(hands-on)*
- 14:15 - 14:30: **Evaluation**

Welcome!

- 12:30 - 12:45: **Introduction** *(presentation)*
- 12:45 - 13:30: **Part 1: Hello, world!** *(guided hands-on)*
- 13:30 - 13:45: **Break**
- 13:45 - 14:15: **Part 2: A workflow for Disaster Tweets** *(hands-on)*
- 14:15 - 14:30: **Evaluation**

Background

Introducing… Tim Müller

- **23** years old
- Now: **Scientific Programmer** for the EPI Project
- Before: Bachelor **Artificial Intelligence** at University of Amsterdam
- Master **Computer Science, Security Track**, UvA/VU (not completed)

- Developer in charge of EPI Framework
  - Combine various parts developed by PhD students
  - Deploy framework at hospitals

Introducing… EPI Project

- EPI -> **Enabling Personalized Interventions**
- Goal: introduction of *Digital Health Twins*
- Various aspects investigated
  - Statistical learning and hypothesis testing
  - Distributed machine learning
  - Privacy-preserving machine learning
  - Automated policy interpretation and enforcement
  - **General-purpose data-sharing framework**

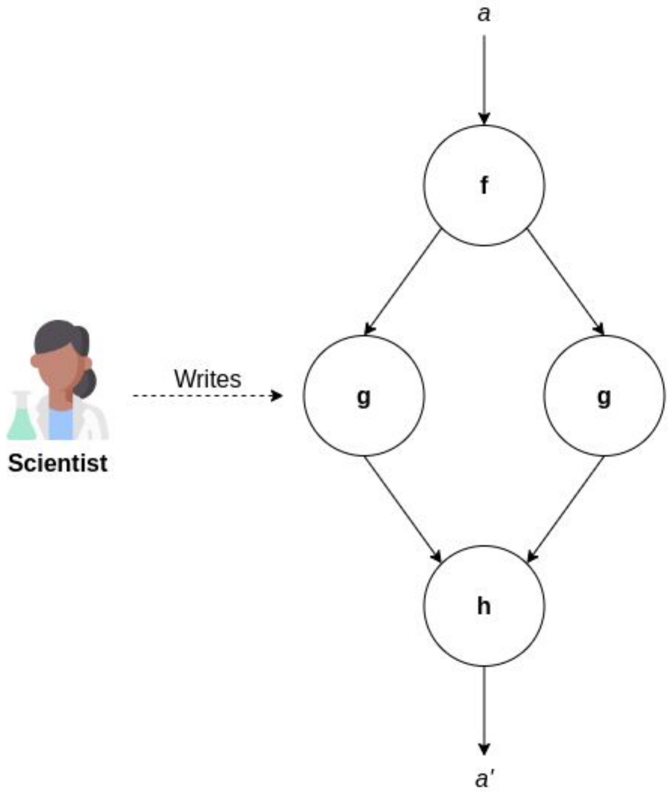
Introducing… EPI Framework

- **EPI Framework** -> general purpose data sharing framework
- Actually, **distributed workflow execution system**
  - i.e., typically, algorithm-to-data
- Focus on **healthcare**
  - Challenges: private datasets -> private policies, autonomous domains
- Built on **BRANE**
  - Focus on separation of concerns and flexibility (hence framework)
- Integrates other technologies
  - BFC Framework
  - Policy reasoners (eFLINT)

EPI Framework Overview

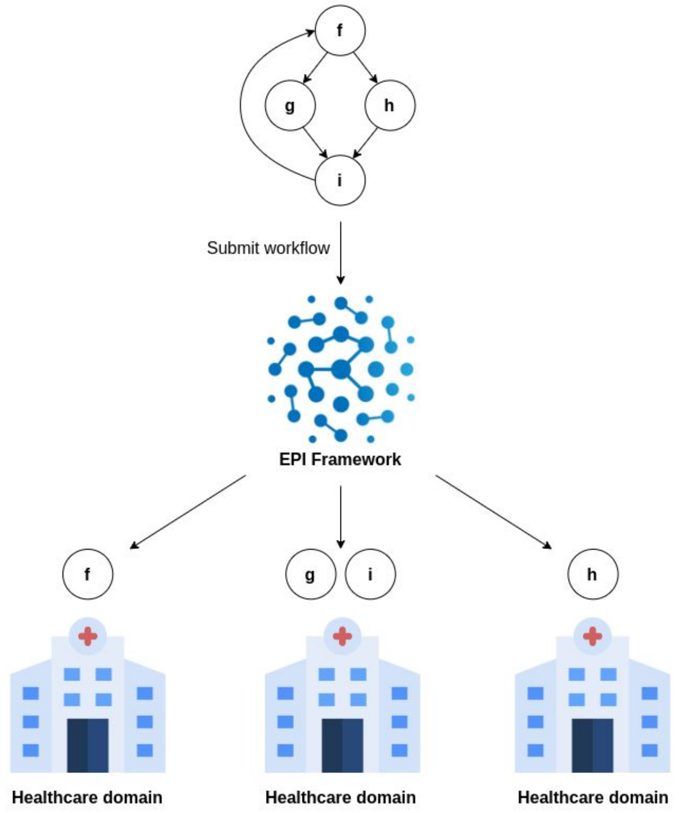
Workflows

- **High-level**, **distributed** programs
  - Visualizable as graphs
  - Nodes are **tasks** or **functions**
  - Edges are some **dependency** (typically data)
- Workflow system's job to **fill in details**
  - This act we call *planning*
- Most importantly, **locations** and **ordering** is left to system
  - But the more, the better



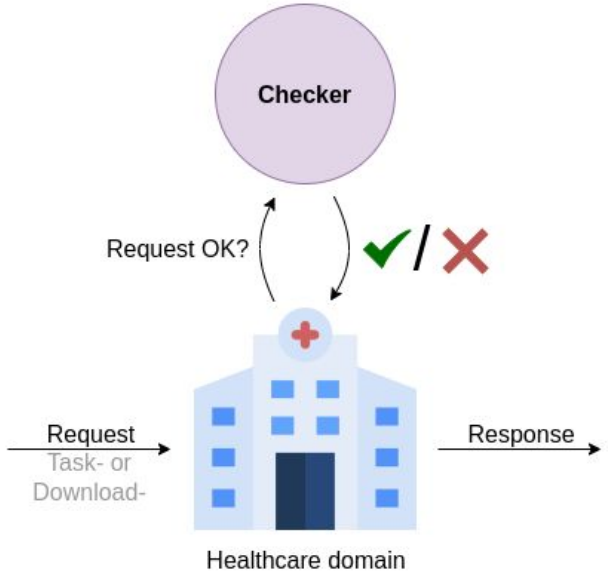
Workflow execution systems

- Typically some kind of **federated system**
- An **orchestrator** is the central part
  - Plans workflows
  - Executes plans
- A **worker** is the local or distributed part
  - Executes individual tasks



EPI Framework

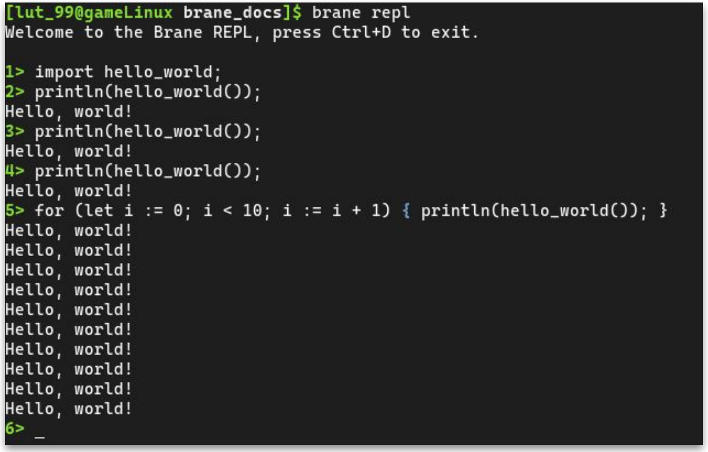
- Orchestrator -> **central node**
- Workers -> **worker nodes**
- Task peculiarities:
  - Tasks are **containerized**
  - Multiple tasks in one container, called a **package**
- Data peculiarities:
  - Workers also host **datasets** or **assets**
  - Workers have **checkers**
  - Checkers enforce **policies**
  - Policies **limit** execution and data access



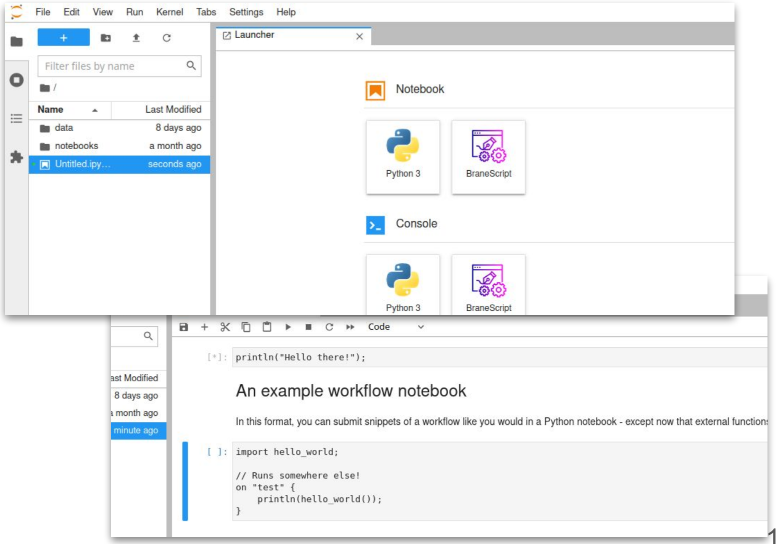
EPI Framework as a user

Client

- A running system is an **instance**
- A **client** is used to submit workflows / upload packages to specific **instance**
- Current client: **`brane`** executable (CLI tool)
  - JupyterLab integration work-in-progress
- **Local execution** also possible



```
[lut_99@gameLinux brane_docs]$ brane repl
Welcome to the Brane REPL, press Ctrl+D to exit.

1> import hello_world;
2> println(hello_world());
Hello, world!
3> println(hello_world());
Hello, world!
4> println(hello_world());
Hello, world!
5> for (let i := 0; i < 10; i := i + 1) { println(hello_world()); }
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
6> _
```



```
[*]: println("Hello there!");
```

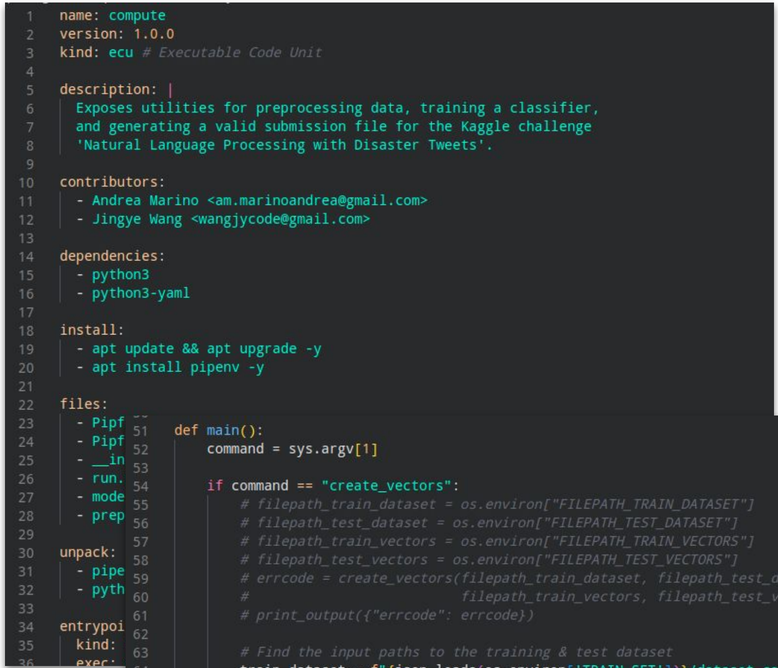
An example workflow notebook

In this format, you can submit snippets of a workflow like you would in a Python notebook - except now that external function

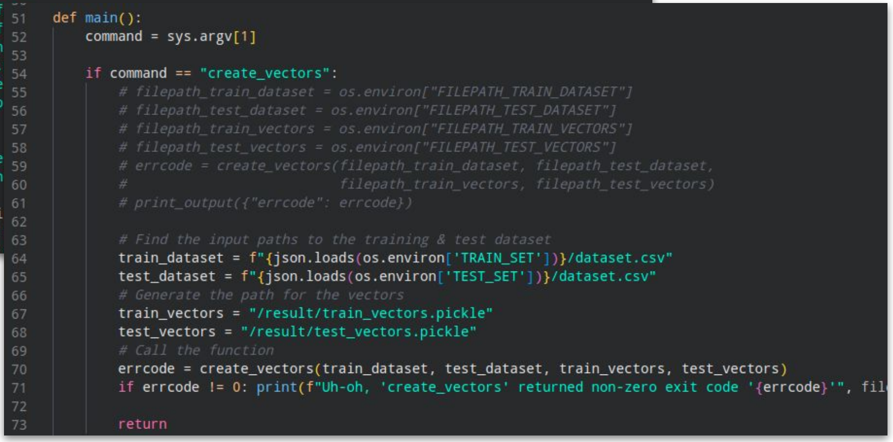
```
[ ]: import hello_world;

     // Runs somewhere else!
     on "test" {
         println(hello_world());
     }
```

Packages

- Role: **Software Engineer**
- Packages are **containerized**
- Language: **any**!
- Metadata specified in **YAML**
- But there is **specific interface**
    - Task selection with **arguments**
    - Input via **environment variables** (JSON)
    - Output via **stdout** (YAML)
    - Data is provided as a **read-only file**



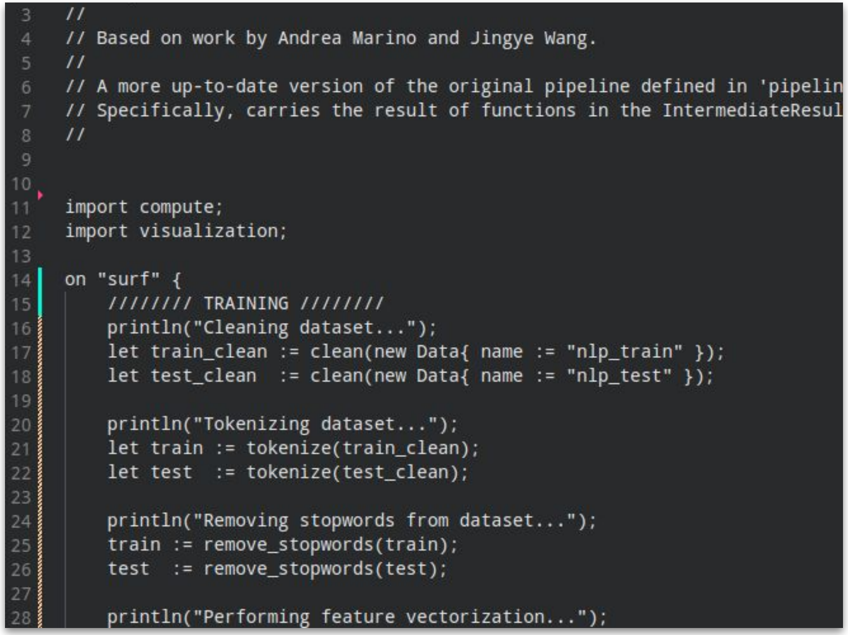
```yaml
name: compute
version: 1.0.0
kind: ecu # Executable Code Unit

description: |
  Exposes utilities for preprocessing data, training a classifier,
  and generating a valid submission file for the Kaggle challenge
  'Natural Language Processing with Disaster Tweets'.

contributors:
  - Andrea Marino <am.marinoandrea@gmail.com>
  - Jingye Wang <wangjycode@gmail.com>

dependencies:
  - python3
  - python3-yaml

install:
  - apt update && apt upgrade -y
  - apt install pipenv -y

files:
  - Pipf
  - Pipf
  - __in
  - run.
  - mode
  - prep

unpack:
  - pipe
  - pyth

entrypoi
  kind:
  exec:
```



```python
def main():
    command = sys.argv[1]

    if command == "create_vectors":
        # filepath_train_dataset = os.environ["FILEPATH_TRAIN_DATASET"]
        # filepath_test_dataset = os.environ["FILEPATH_TEST_DATASET"]
        # filepath_train_vectors = os.environ["FILEPATH_TRAIN_VECTORS"]
        # filepath_test_vectors = os.environ["FILEPATH_TEST_VECTORS"]
        # errcode = create_vectors(filepath_train_dataset, filepath_test_dataset,
        #                          filepath_train_vectors, filepath_test_vectors)
        # print_output({"errcode": errcode})

        # Find the input paths to the training & test dataset
        train_dataset = f"{json.loads(os.environ['TRAIN_SET'])}/dataset.csv"
        test_dataset = f"{json.loads(os.environ['TEST_SET'])}/dataset.csv"
        # Generate the path for the vectors
        train_vectors = "/result/train_vectors.pickle"
        test_vectors = "/result/test_vectors.pickle"
        # Call the function
        errcode = create_vectors(train_dataset, test_dataset, train_vectors, test_vectors)
        if errcode != 0: print(f"Uh-oh, 'create_vectors' returned non-zero exit code '{errcode}'", fil

        return
```

Workflows

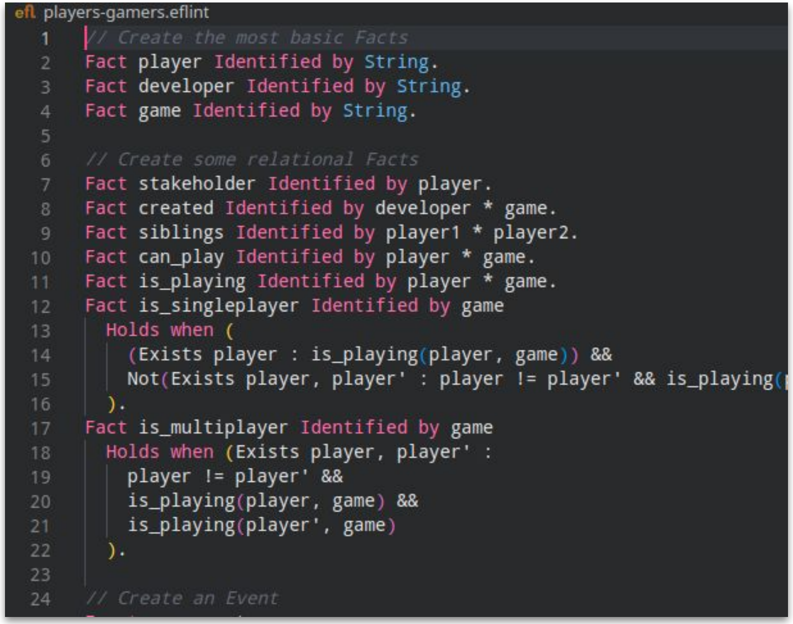
- Role: **Scientist**
- Front-end is with **custom Domain-Specific Language** (DSL)
    - **BraneScript**
    - Bakery (WIP)
- Compiles to common Intermediate Representation (IR)
- **Script-like control flow** statements (if, for, while, parallel)
- Can manually specify locations (on-structs)
- Other languages planned: **OpenAPI, CWL**



```
//
// Based on work by Andrea Marino and Jingye Wang.
//
// A more up-to-date version of the original pipeline defined in 'pipelin
// Specifically, carries the result of functions in the IntermediateResul
//


import compute;
import visualization;

on "surf" {
    //////// TRAINING ////////
    println("Cleaning dataset...");
    let train_clean := clean(new Data{ name := "nlp_train" });
    let test_clean  := clean(new Data{ name := "nlp_test" });

    println("Tokenizing dataset...");
    let train := tokenize(train_clean);
    let test  := tokenize(test_clean);

    println("Removing stopwords from dataset...");
    train := remove_stopwords(train);
    test  := remove_stopwords(test);

    println("Performing feature vectorization...");
```

Policies (out-of-scope)

- Role: **Policy Expert**
- Might vary per hospital
- Behind service, so **any language**!
- Possible choices:
  - XACML
  - eFLINT
  - Python
  - Prolog
  - …
- (Not yet fully implemented)

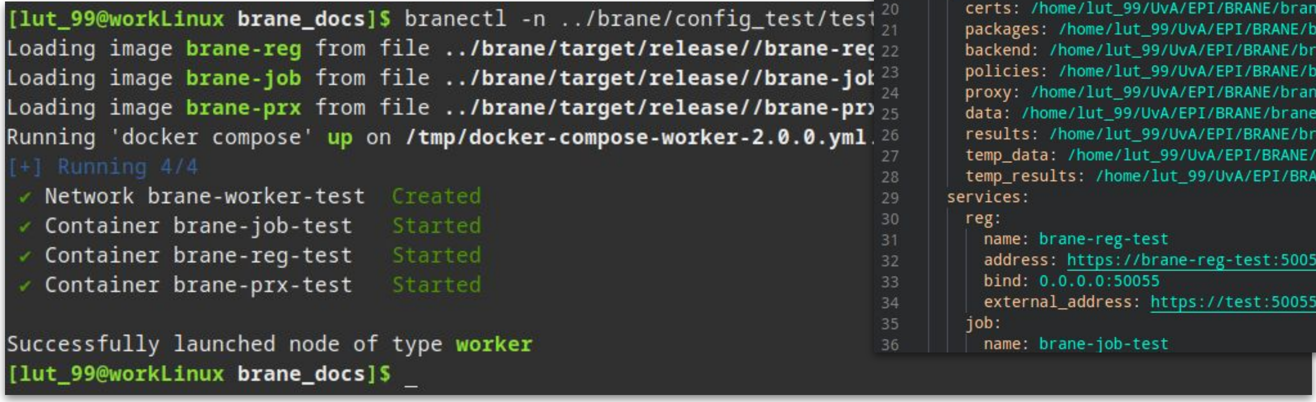


players-gamers.eflint

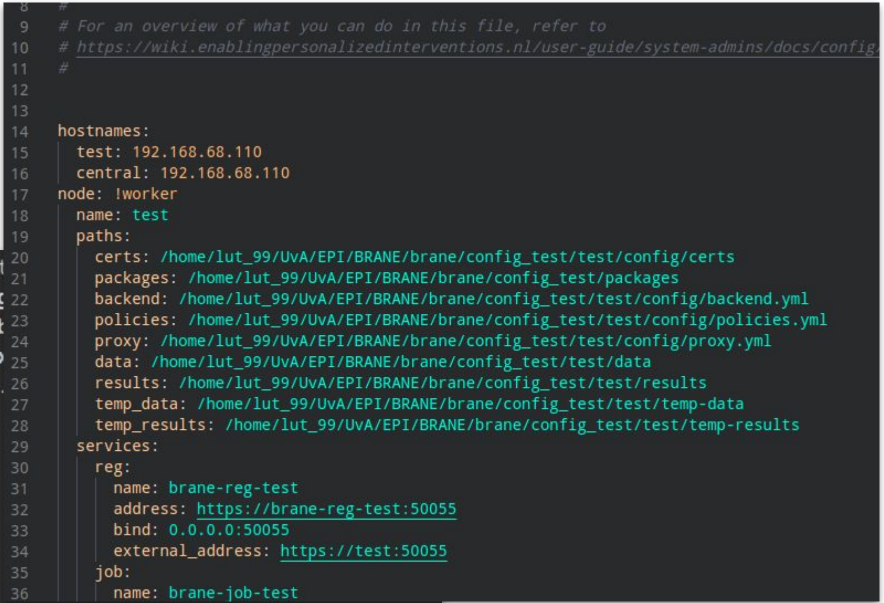
```
// Create the most basic Facts
Fact player Identified by String.
Fact developer Identified by String.
Fact game Identified by String.

// Create some relational Facts
Fact stakeholder Identified by player.
Fact created Identified by developer * game.
Fact siblings Identified by player1 * player2.
Fact can_play Identified by player * game.
Fact is_playing Identified by player * game.
Fact is_singleplayer Identified by game
  Holds when (
    (Exists player : is_playing(player, game)) &&
    Not(Exists player, player' : player != player' && is_playing(
  ).
Fact is_multiplayer Identified by game
  Holds when (Exists player, player' :
    player != player' &&
    is_playing(player, game) &&
    is_playing(player', game)
  ).

// Create an Event
```

Administration (out-of-scope)

- Role: **System Administrator**
- **`branectl`** CLI tool
- Various **configuration files**
    - In YAML, mostly



```
[lut_99@workLinux brane_docs]$ branectl -n ../brane/config_test/test
Loading image brane-reg from file ../brane/target/release//brane-reg
Loading image brane-job from file ../brane/target/release//brane-job
Loading image brane-prx from file ../brane/target/release//brane-prx
Running 'docker compose' up on /tmp/docker-compose-worker-2.0.0.yml.
[+] Running 4/4
 ✔ Network brane-worker-test   Created
 ✔ Container brane-job-test    Started
 ✔ Container brane-reg-test    Started
 ✔ Container brane-prx-test    Started

Successfully launched node of type worker
[lut_99@workLinux brane_docs]$ _
```



```
#
# For an overview of what you can do in this file, refer to
# https://wiki.enablingpersonalizedinterventions.nl/user-guide/system-admins/docs/config/
#


hostnames:
  test: 192.168.68.110
  central: 192.168.68.110
node: !worker
  name: test
  paths:
    certs: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/config/certs
    packages: /home/lut_99/UvA/EPI/BRANE/brane/config_test/packages
    backend: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/config/backend.yml
    policies: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/config/policies.yml
    proxy: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/config/proxy.yml
    data: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/data
    results: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/results
    temp_data: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/temp-data
    temp_results: /home/lut_99/UvA/EPI/BRANE/brane/config_test/test/temp-results
  services:
    reg:
      name: brane-reg-test
      address: https://brane-reg-test:50055
      bind: 0.0.0.0:50055
      external_address: https://test:50055
    job:
      name: brane-job-test
```

As a user - Summary

- **Different roles** have **different tools**
    - Hopefully familiar to that role
- Lots of **flexibility** for users
    - Different languages, different interfaces (although not all of them are implemented)
- In this tutorial, you will be a **scientist** and a **software engineer**

Part 1: Hello, world!

Objective

- Write your first **Hello, world!-package**
- See the steps at https://wiki.enablingpersonalizedinterventions.nl/user-guide
    - Bottom-left, scroll down to “Tutorials”, then “Part 1: Hello, world!”
    - Or see: https://tinyurl.com/2xft4cp3
- I’ll go through it on the board

Tip: Use the `jaarbeurshotspot` WiFi

Note: The framework is experimental, so unfortunately, expect rough edges

Break

Part 2: A workflow for Disaster Tweets

Objective

- Write a workflow for a package that **classifies disaster tweets**
- See the steps at https://wiki.enablingpersonalizedinterventions.nl/user-guide
  - Bottom-left, scroll down to “Tutorials”, then “Part 2: A workflow for Disaster Tweets”
  - Or see: https://tinyurl.com/mtv24wwp
- Try to go through the steps yourself!
  - Or just play with the framework :)

Let me know if you have questions or need help!

Evaluation

Framework experience?

- Did you find the package design intuitive?
  - Was it too complex?
  - Was the data format (YAML) / language unpleasant?
  - Did running the package locally make sense?
- Did you find workflow writing intuitive?
  - Yes to DSL, no to DSL?
  - DSL intuitive?
  - Is logging in to remote instances clear?
  - Impression about performance?

Tutorial experience?

- Is the material clear enough?
- Were there any unsolvable bugs?
- Did you have enough time?
- Did you like the guided hands-on?
- Did you like the individual hands-on?
- Do you like the topic (scientist/software engineer)?

BRANE

Tim Müller (t.muller@uva.nl)

enablingpersonalizedinterventions.nl

github.com/epi-project/brane

wiki.enablingpersonalizedinterventions.nl (WIP)

The icons (not logos) in this presentation are from Flaticon.com