



Mélodium

Améliorer les performances et réduire la consommation de la CI/CD
– Pour **France DevOps**

<https://melodium.tech>



Quentin Vignaud

Créateur de la technologie Mélodium, ancien de Doctolib

M. Sc. Sciences Informatiques – UQAM (Canada)
Ingénieur Développement Logiciel – CESI (France)

« Détection et classification des éléments d'une piste audio musicale » – Thèse UQAM 2020

Sommaire

01

Mélodium ?

02

**La CI/CD et son
optimisation**

03

**Implémentation
concrète**

04

**Intégration et
compatibilité**

05

La suite

Mélodium Késako ?



Historique Rapide

Créé en laboratoire pour :

- Analyse audio
- Run agnostique
- Gestion de données massives
- Entraînement de modèles de réseaux de neurones

Mélodium Késako ?



Langage

Spécification textuelle et syntaxe définie.



Graphes

Définition de programmes sous forme de graphes orientés.



Moteur d'orchestration

Gestion de tâches, threads, processus, et machines.



Gestion de flux

Transmet et répartit des flux de données et signaux.



Formalisme

Structure le développement, vérifie la validité des actions.



Libre

Logiciel libre sous EUPL
*European Union Public
Licence*

Traitements, flux, et connexions

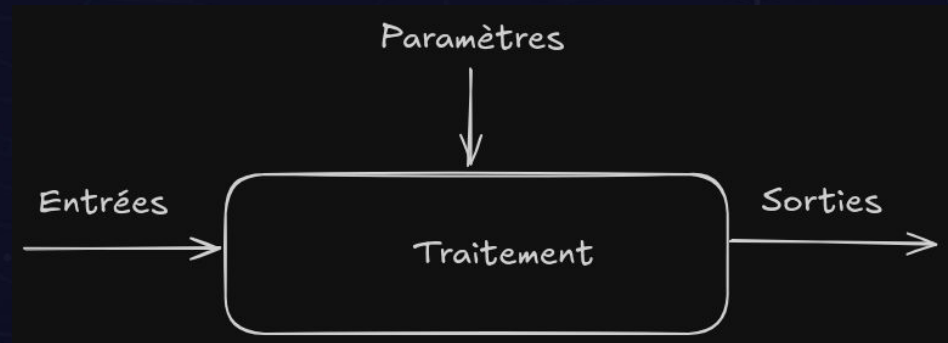


Un traitement

- Paramètres
- Entrées / Inputs
- Sorties / Outputs

En interne :

- Traitements enfants
- Connexions



Traitements, flux, et connexions



```
1 use fs/local::writeLocal
2 use std/text/convert/string::toUtf8
3
4 treatment writeText(
5     | var filename: string
6 )
7 input text: Stream<string>
8 output written_bytes: Stream<u128>
9 {
10     toUtf8()
11     writeLocal(
12         | path = filename
13     )
14
15     Self.text -> toUtf8.text, encoded -> writeLocal.data, amount -> Self.written_bytes
16 }
```

Traitements, flux, et connexions

Unsigned integers	Signed integers	Floating-point numbers	Text	Logic
u8	i8	f32	char	byte
u16	i16	f64	string	bool
u32	i32			void
u64	i64			
u128	i128			

- + Types de conteneurs **Vec**, **Option**, **Map**, etc.
- + Types de données construits **Json**, **Command**, etc.
- + Support de génériques
- + Support de traits

Fonctions, Modèles, Contextes...

Livre : <https://doc.melodium.tech/book/en/>

Connexions

Stream

→ Flux continu de données

Block

→ Unique élément transmis

Paramètres

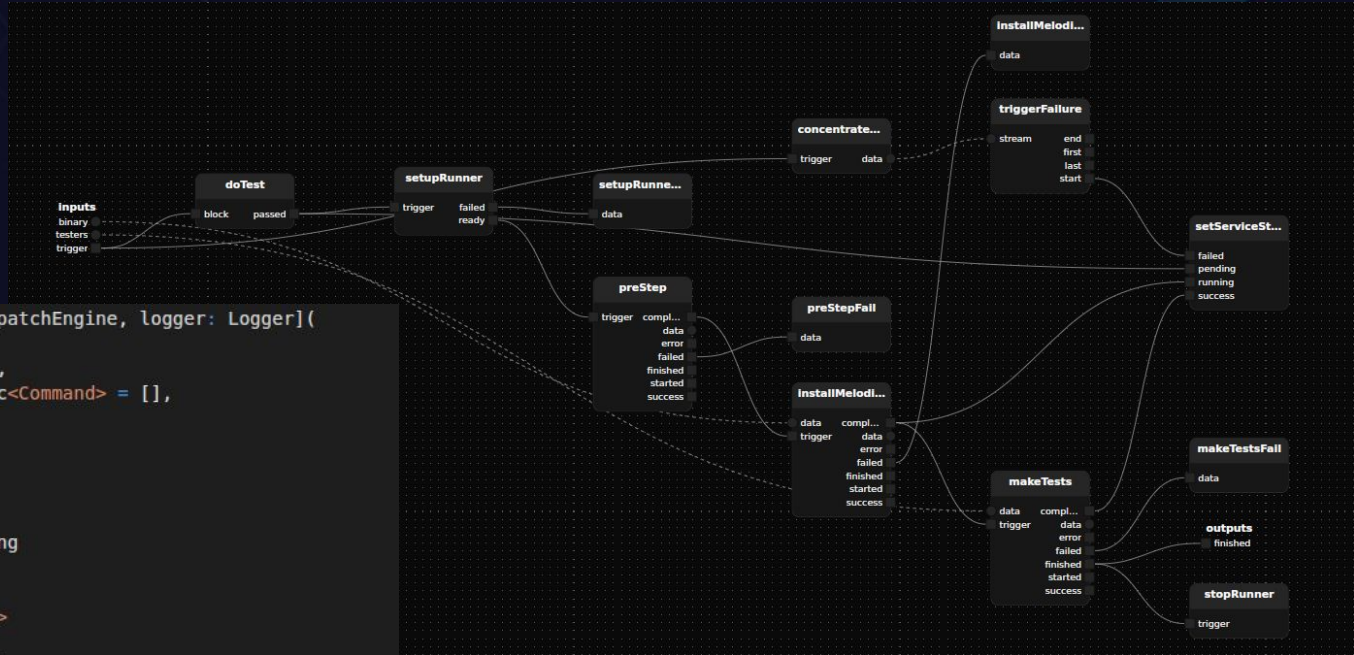
Var

→ Valeur possiblement différente pour chaque piste

Const

→ Valeur constante pour l'instanciation donnée

Traitements, flux, et connexions



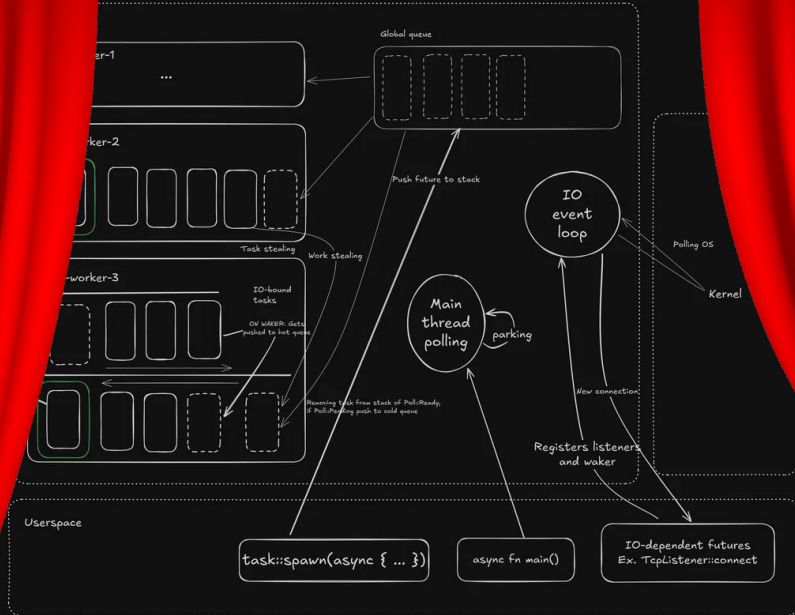
```
31 treatment test[cicd: CicdDispatchEngine, logger: Logger](
32     var arch: Arch,
33     var pipeline: string,
34     var pre_commands: Vec<Command> = [],
35     var project: string,
36     var ref: string,
37     var sha: string,
38     var target: string,
39     var test: bool,
40     var test_image: string
41 )
42 input binary: Stream<byte>
43 input testers: Stream<byte>
44 input trigger: Block<void>
45 output finished: Block<void>
46 model failureConcentrator: Concentrator()
47 model runner: CicdRunnerEngine()
48 {
49     /* ... */
50 }
```

Comment ça fonctionne ?



Un moteur asynchrone

- Écrit en Rust
- Usage de futures asynchrones
- Faible empreinte mémoire
- Réactivité au système
- Concurrence collaborative



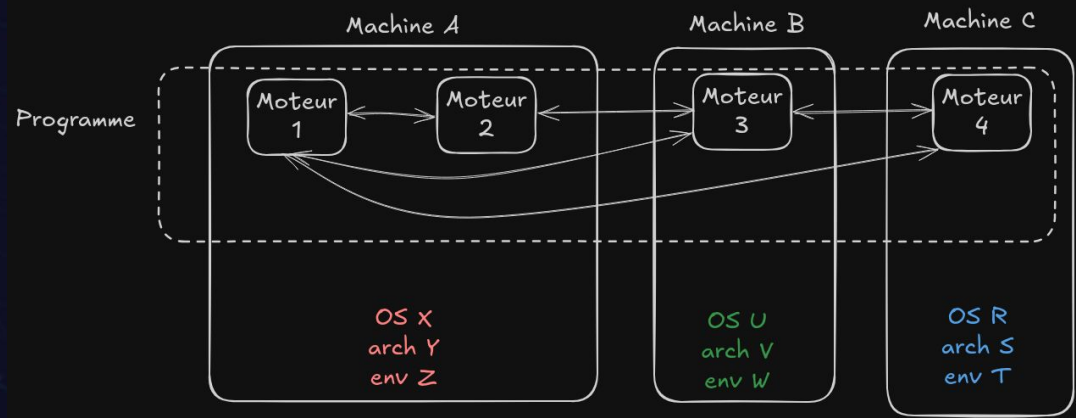
Un programme pour les contrôler tous



Un programme sur plusieurs moteurs

Répartition des traitements et flux au travers de plusieurs moteurs situés sur diverses machines.

- OS différents
- Architectures différentes
- Un seul programme





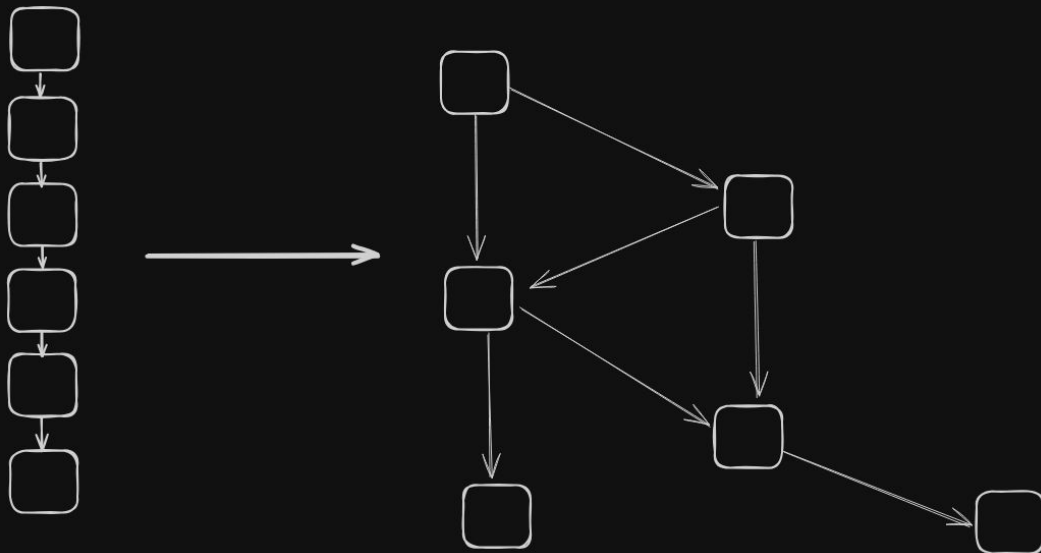
Optimiser la CI/CD 02

Des listes qui deviennent graphes



Délinéariser la CI/CD

Organiser un pipeline en graphe plutôt qu'en liste.

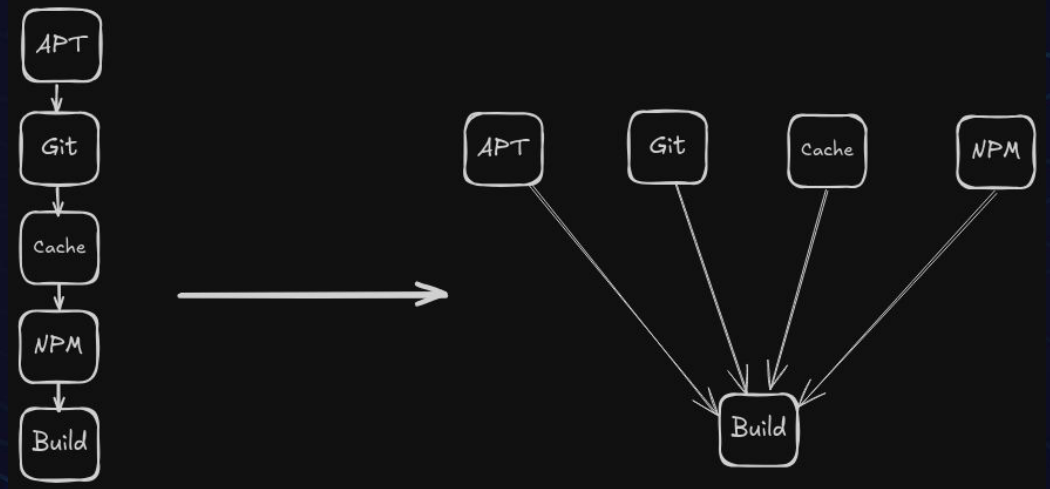


Exemple : Parallélisation de commandes



Étapes préparatoires

- apt-get install
- git pull
- cache pull
- npm install
- build



Ordre imposé vs. enchaînement nécessaire



Éviter un ordre imposé

- ↳ Start
- ↳ Pre Actions
- ↳ Business Actions
- ↳ Post Actions
- ↳ End
- ↳ Results

Requiert la finalisation d'un pipeline pour obtenir des résultats.

Avoir l'ordre nécessaire

Faire les actions dans l'ordre logique le plus pertinent pour obtenir un résultat exploitable.

Faire au plus court, sans attendre l'accessoire.

Le temps développeur vaut davantage que le temps machine.

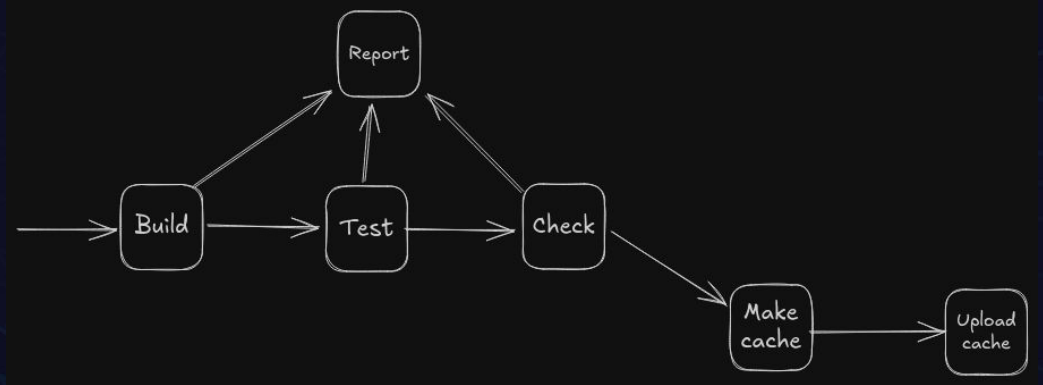
Exemple : Cache build, push, puis résultats



Étapes postérieures

Build
↓
Test
↓
Check
↓
Store
↓
Upload
↓
Report

Attente du pipeline
complet pour avoir les
résultats



- Liberté d'ordonnancement
- Résultats significatifs avant la fin du pipeline

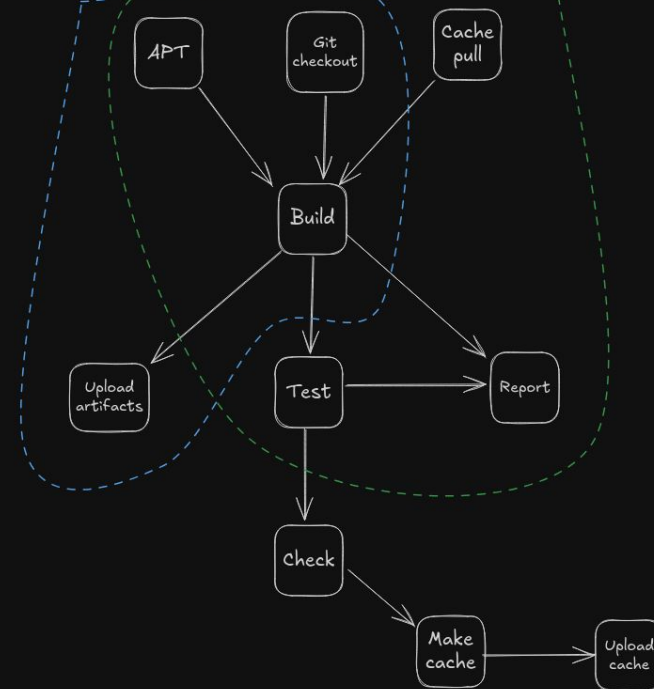
Focus sur la partie critique



Réduction du chemin critique de pipeline au strict nécessaire.

Production Release

Developer Push





**Optimiser *un* pipeline
c'est bien...**

**...optimiser *les* pipelines
c'est mieux !**



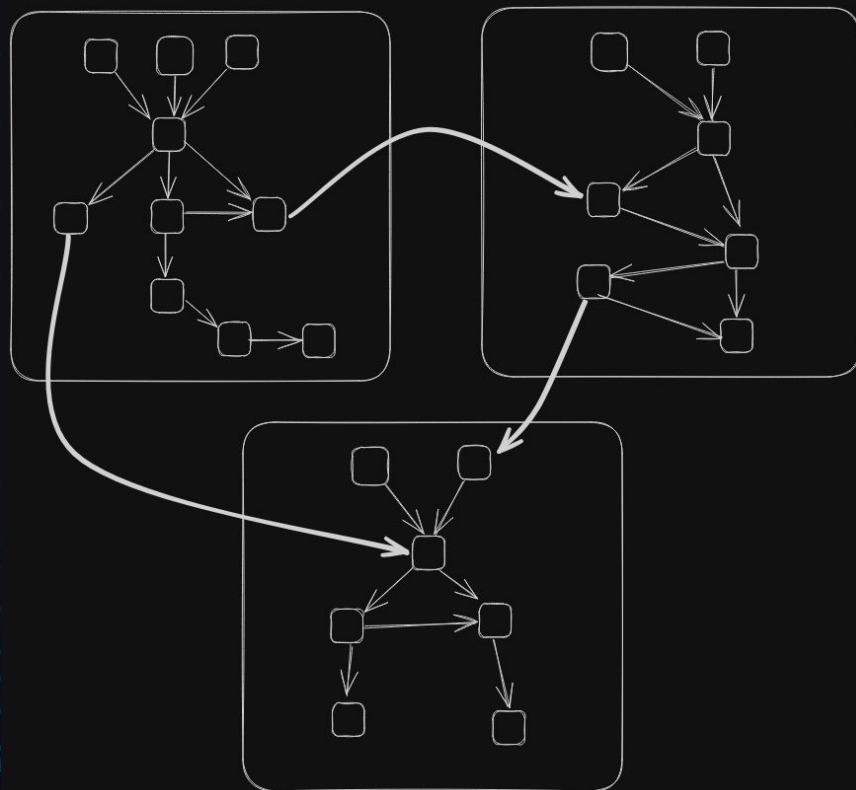
Connexions entre pipelines



Les Push/Store/Pull d'artéfacts deviennent des connexions entre pipelines

Autant de connexions que nécessaire

Isolation fonctionnelle des pipelines maintenues

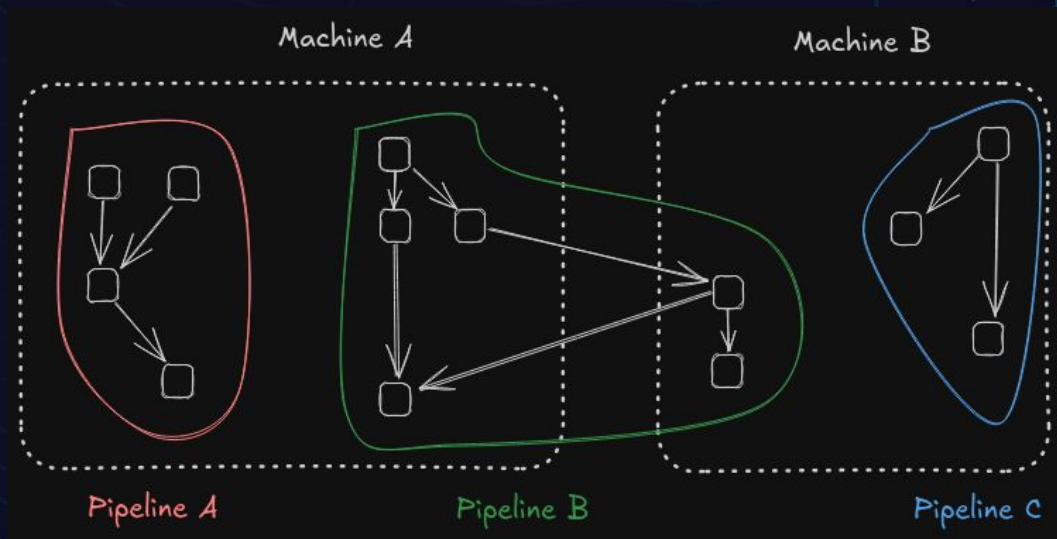


Orchestration de machines



Disruption de la relation 1:1 entre machines et pipelines

- Regroupement de pipelines sur une même machine
- Répartition d'un pipeline sur diverses machines

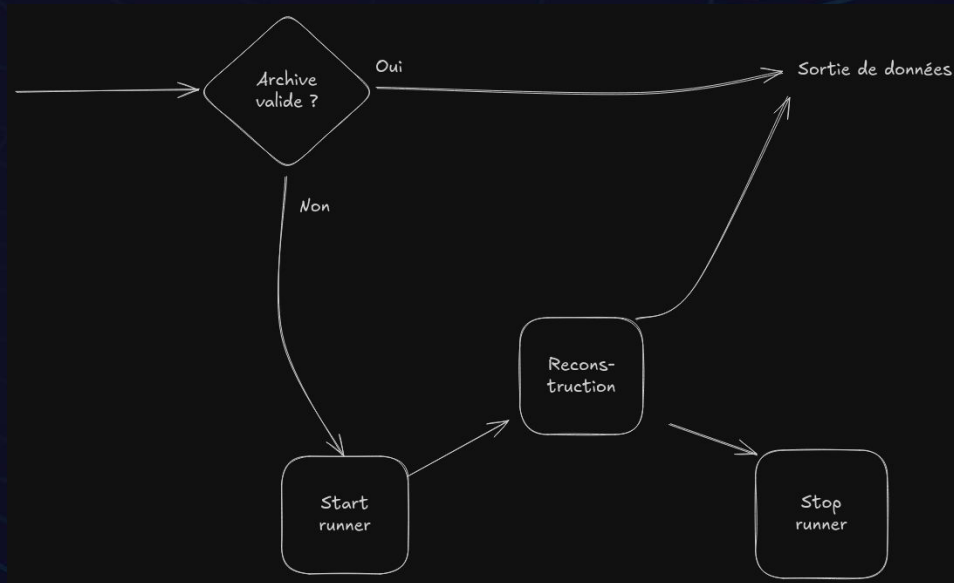


Instanciation au besoin



Machines et conteneurs instanciés au besoin.

- Pas de préconfiguration nécessaire
- Implémentation et exécution souple

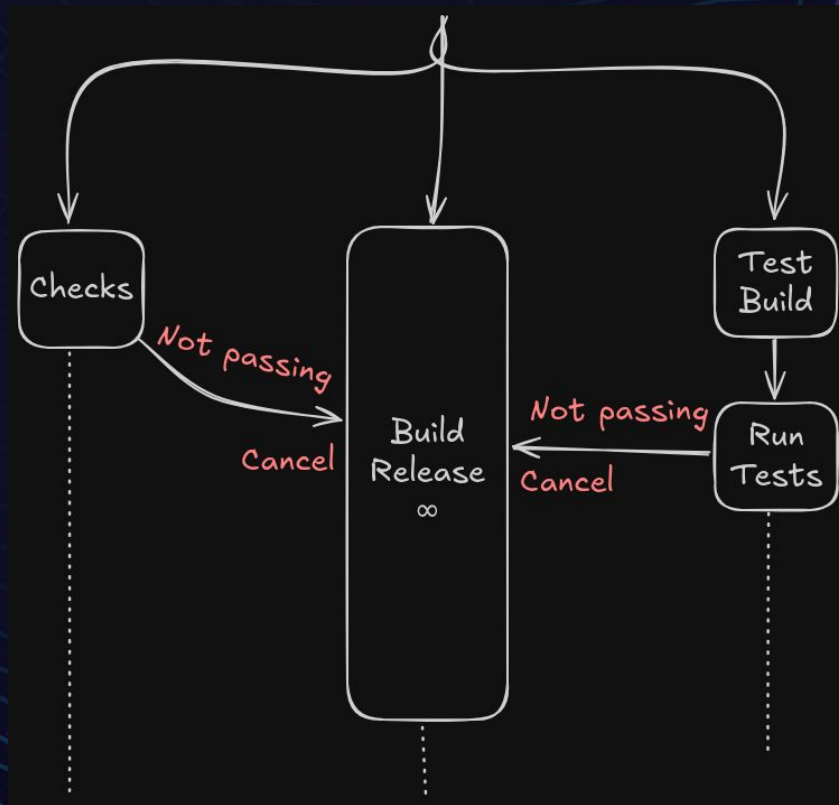


Lancer d'abord, annuler ensuite



Faire les choses à priori

- Réduction du temps front
- Annulation possible





Implémentation

03

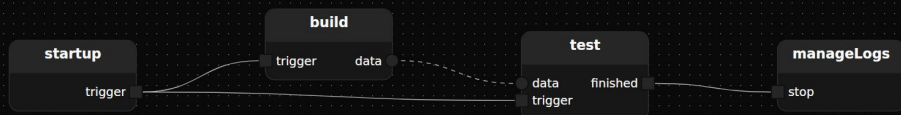
Implémentation



Création d'une CI basique

```
Compo.toml > ...
1 name = "my_cicd"
2 version = "0.1.0"
3
4 [dependencies]
5 cicd = "^0.9.1"
6 distrib = "^0.9.1"
7 fs = "^0.9.1"
8 http = "^0.9.1"
9 log = "^0.9.1"
10 net = "^0.9.1"
11 process = "^0.9.1"
12 std = "^0.9.1"
13 work = "^0.9.1"
14
15 [entrypoints]
16 main = "my_cicd::main"
```

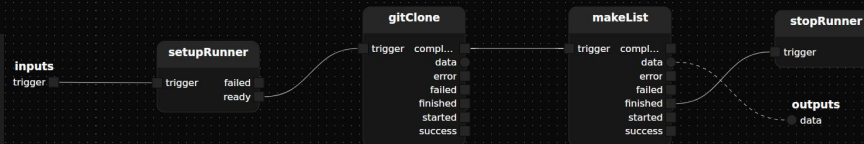
```
$ melodium new --template cicd my_cicd
```



```
20 treatment main(
21     const api_token: string,
22     var output_directory: string,
23     var repository_clone_ref: string,
24     var repository_clone_url: string,
25     const work_location: string = "api"
26 )
27 model cicd: CicdDispatchEngine(api_token = api_token, location = work_location)
28 model logger: Logger()
29 {
30     build[cicd = cicd, logger = logger]{
31         repository_clone_ref = repository_clone_ref,
32         repository_clone_url = repository_clone_url
33     }
34     manageLogs[logger = logger]{
35         output_directory = output_directory
36     }
37     startup()
38     test[cicd = cicd, logger = logger]()
39
40     startup.trigger --> build.trigger, data --> test.data
41     startup.trigger -----> test.trigger, finished --> manageLogs.stop
42 }
43
```

Implémentation

```
23 treatment build[cicd: CicdDispatchEngine, logger: Logger](
24     var repository_clone_ref: string,
25     var repository_clone_url: string
26 )
27 input trigger: Block<void>
28 output data: Stream<byte>
29 model runner: CicdRunnerEngine()
30 {
31     setupRunner[dispatcher = cicd, logger = logger, runner = runner](
32         containers = [[container("ubuntu", 1000, 10, 8000, |amd64(), [|mount("build-result", "/mounted/result" )], "ubuntu:noble", _)],
33         cpu = 10,
34         memory = 500,
35         name = "build",
36         storage = 1000,
37         volumes = [|volume("build-result", 30)]
38     )
39     gitClone: stepOn[logger = logger, runner = runner](
40         commands = [
41             |command("apt-get", [|update]),
42             |command("apt-get", [|install, "-y", "git"]),
43             |command("git", [|config, "--global", "url.https://.insteadof", "git://"]),
44             |command("git", [|clone, "--branch", repository_clone_ref, "--depth", "1", repository_clone_url, "project"])
45         ],
46         environment = |wrap<Environment>{
47             |environment(|map([], "/root", false, false)
48         ),
49         executor_name = "ubuntu",
50         name = "git"
51     )
52     makeList: stepOn[logger = logger, runner = runner](
53         commands = |raw commands{
54             "sh -c \\'ls -l --almost-all | tee files-list.txt\'",
55             "mv files-list.txt /mounted/result/"
56         },
57         environment = |wrap<Environment>{
58             |environment(|map([], "/root/project", false, false)
59         ),
60         executor_name = "ubuntu",
61         name = "list",
62         out_file = "files-list.txt",
63         out_filesystem = "build-result"
64     )
65     stopRunner[runner = runner]()
66 }
67 Self.trigger -> setupRunner.trigger,ready -> gitClone.trigger,completed -> makeList.trigger,finished -> stopRunner.trigger
68 makeList.data -> Self.data
69 }
```



Équivalent d'un job

- Définition de traitements
- Branchement de connexions
 - ◆ Connexions de déclencheurs
 - ◆ Connexions de données
- Démarrage et arrêt de runner à la volée

Implémentation



CicdDispatchEngin

e

```
25 treatment main(  
26     const api_token: string,  
27     const work_location: string = "api"  
28 )  
29 model cicd: CicdDispatchEngine(api_token = api_token, location = work_location)  
30 model logger: Logger()
```

- “api” : Localisation du travail par API distante
 - ◆ API sur SaaS, Kubernetes, Cloud provider
 - ◆ API auto-hébergée
- “compose” : Dispatch en local avec podman ou docker compose
 - ◆ Test de CI sur sa propre machine possible

Implémentation



setupRunner

- Instancie un runner grâce au CicdDispatchEngine
- Définit les caractéristiques pour le moteur distant
- Configure des conteneurs
- Configure des volumes

setupRunner

trigger

failed

ready

```
13 treatment runnerBuild[cicd: CicdDispatchEngine, logger: Logger, runner: CicdRunnerEngine](
14     var name: string,
15     var on: Arch,
16     var rust_memory: u32 = 4000,
17     var rust_storage: u32 = 5000
18 )
19 input trigger: Block<void>
20 output failed: Block<void>
21 output ready: Block<void>
22 {
23     setupRunner[dispatcher = cicd, logger = logger, runner = runner](
24         containers = [|container("rust", rust_memory, 2000, rust_storage, on, [|mount("compilation-result", "/mounted/result"), "rust:bullseye", _]),
25         cpu = 1000,
26         memory = 500,
27         name = name,
28         stop_on_failure = false,
29         storage = 1300,
30         volumes = [|volume("compilation-result", 1000)]
31     )
32
33     Self.trigger -> setupRunner.trigger
34     setupRunner.ready -> Self.ready
35     setupRunner.failed -> Self.failed
36 }
37
```

<https://doc.melodium.tech/latest/en/cicd/runners/setupRunner.html>

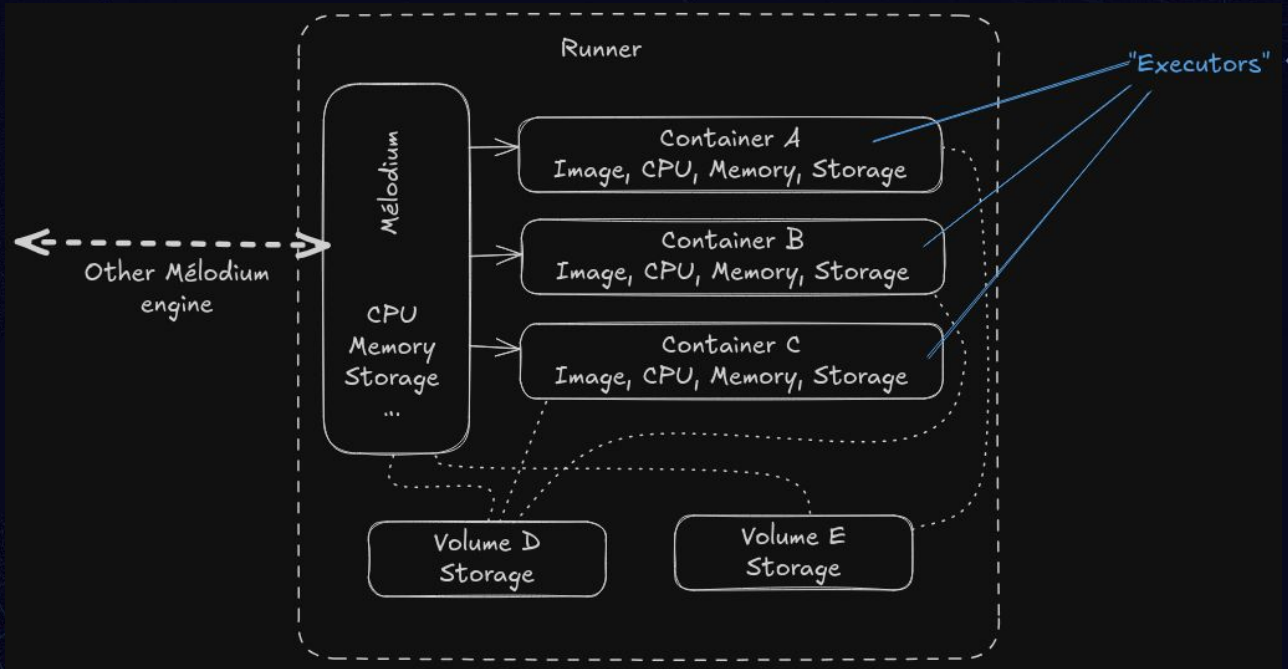
Implémentation



setupRunner

- Conteneurs
- VM
- Machine Physique
- Job K8S
- Compose

...



Implémentation



stepOn

- Exécute les commandes
- Sors la donnée de l'emplacement indiqué out_*

```
55 compilation: stepOn[logger = logger, runner = runner](
56   commands = [
57     |command("rustc", ["--version"]),
58     |command("cargo", ["--version"]),
59     |command("cargo", ["build", "--package", "melodium", "--target", target]),
60     |command("mv", [|format("target/{target}/{kind}/melodium{ext}",
61       |map([
62         |entry("target", target),
63         |entry("kind", |condition<string>(release, "release", "debug")),
64         |entry("ext", |condition<string>(|contains(target, "windows"), ".exe", "")
65       ]))
66     ], "/mounted/result/melodium"]
67   )
68 ],
69   environment = |wrap<Environment>(
70     |environment(environment_variables, "/work/melodium", true, false)
71   ),
72   executor_name = "rust",
73   name = |format("{target} compilation", |entry("target", target)),
74   out_file = "melodium",
75   out_filesystem = "compilation-result",
76   stop_on_failure = true
77 )
```

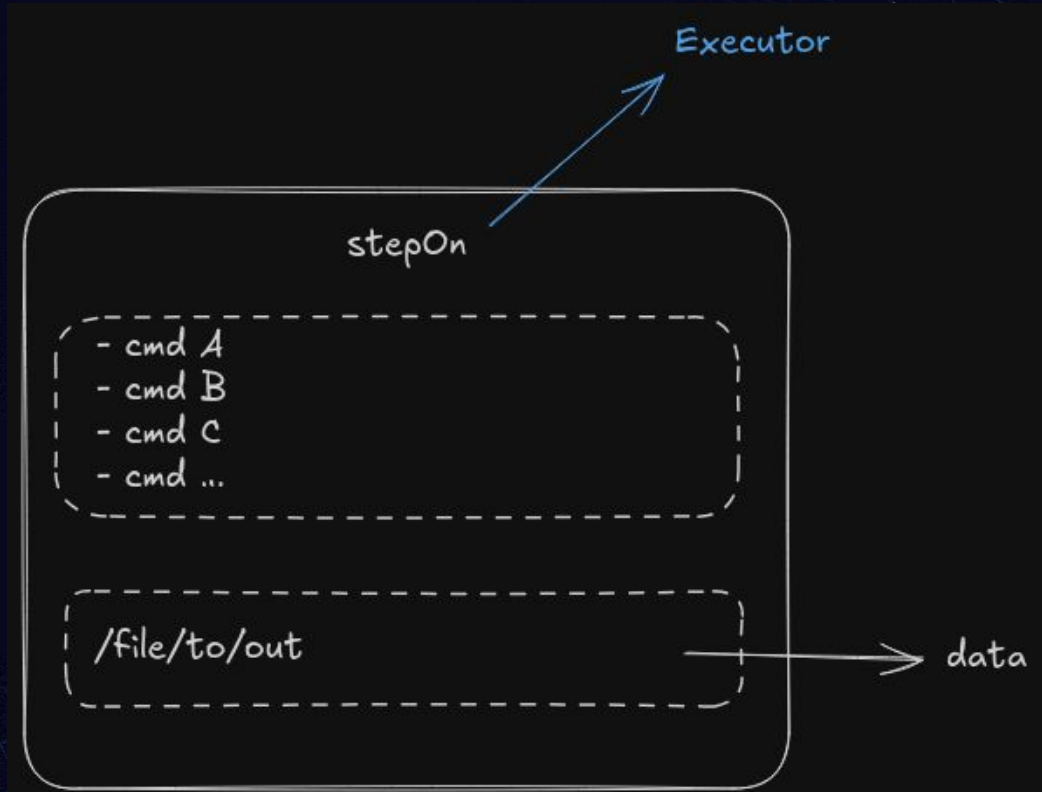


Implémentation



stepOn

- Exécute les commandes
- Sors la donnée de l'emplacement indiqué out_*



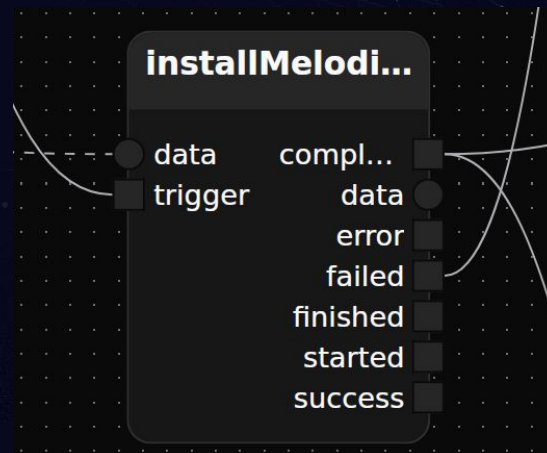
Implémentation



stepOnWithInput

Identique à stepOn, mais :

- Inscrit de la donnée à l'emplacement indiqué par `in_*`



```
57 installMelodium: stepOnWithInput[logger = logger, runner = runner](
58     commands = [
59         |command("mv", ["/mounted/binaries/melodium", "/usr/local/bin/melodium"]),
60         |command("chmod", ["+x", "/usr/local/bin/melodium"])
61     ],
62     executor_name = "testers",
63     in_file = "melodium",
64     in_filesystem = "binaries",
65     name = |format("test:{target}", |entry("target", target))
66 )
```

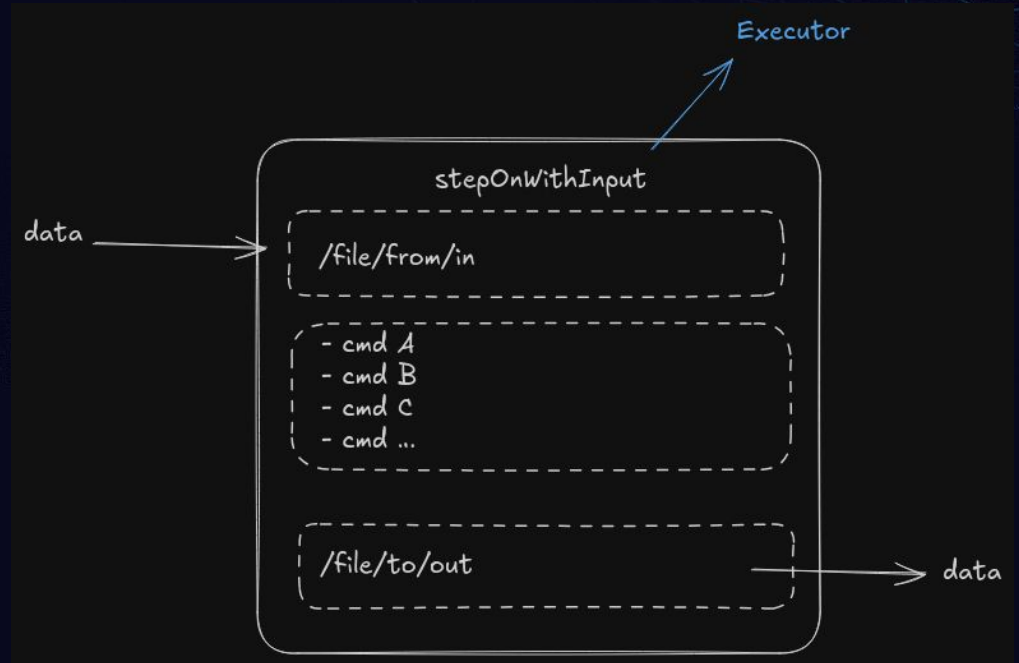
Implémentation



stepOnWithInput

Identique à stepOn, mais :

- Inscrit de la donnée à l'emplacement indiqué par `in_*`

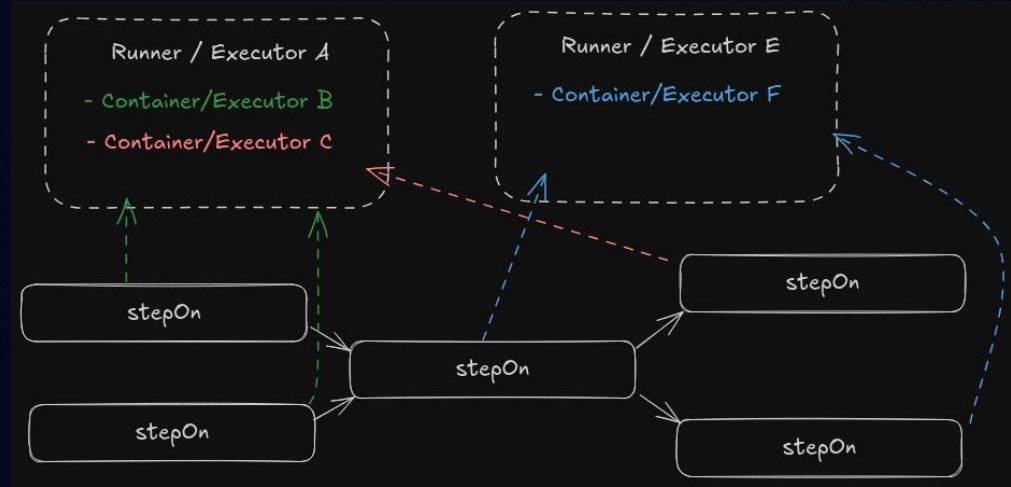


Implémentation



Exécution

- Plusieurs steps peuvent avoir lieu simultanément
- ◆ Sur des runners différents
 - ◆ Sur un même runner
 - ◆ Sur un même executor



Implémentation



Librairie Standard `std`

`std/flow`

Gestion et manipulation des flux.

`std/ops`

Opération sur les données.

`std/*`

`conv/`
`data/`
`text/`
`types/`
...

Autres librairies et modules :
`fs`, `http`, `json`, `process`, `net`, `regex`,
`sql`...

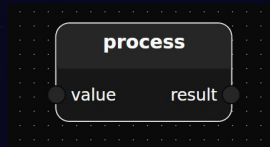
<https://doc.melodium.tech/latest/en>

Implémentation – Bonus



JavaScriptEngine

```
32 treatment jsTreatment()
33   input data: Stream<Json>
34   output data: Stream<Json>
35   model js_engine: JavaScriptEngine(code="function greet(someone) {
36     return {
37       response: `Hello ${someone}!`,
38       ps: \"Thanks for contacting us\"}
39     }")
40 {
41   process[engine = js_engine](code = "greet(value)")
42   unwrapOr<Json>(default = |null())
43
44   Self.data -> process.value
45   process.result -> unwrapOr.option, value -> Self.data
46 }
```



Permet l'usage de Javascript.

- Code JS exécuté pour chaque valeur en transit
- Entrée et sortie sous format JSON

Implémentation de procédures et fonctions utilitaires, à toutes fins utiles.

Run

```
$ melodium run .melodium-ci/Compo.toml myEntrypoint --output_directory="logs/"
```



Run

- Run par point d'entrée, les paramètres du traitement d'entrée sont des arguments CLI

```
[entrypoints]
main = "ci_cd::main"
regenerate_image = "ci_cd/images::generate"
daily = "ci_cd::dailyTests"
```



Check

- Check possible, sans run
- ```
$ melodium check .melodium-ci/Compo.toml
```
- Principe *fail-fast* : l'intégralité du programme est vérifié avant toute exécution effective. Pas d'échec après le lancement « à la découverte de la ligne ».

# Quicksheet



## Runners

- ◆ CidRunnerEngine
  - ⇒ setupRunner
  - ⇒ stopRunner



## Exécution

- ⇒ stepOn
- ⇒ stepOnWithInput
- ⇒ localStep
- ⇒ simpleStep



## Gestion de logs

- ◆ Logger
  - ⇒ manageLogs



## Dispatch

- ◆ CidDispatchEngine
  - API
  - Compose (local)



## JS

- ◆ JavaScriptEngine
  - ⇒ process



## Référence

- [doc.melodium.tech/latest/en](https://doc.melodium.tech/latest/en)
- [doc.melodium.tech/book/en](https://doc.melodium.tech/book/en)



# Intégration

0  
4

# Intégration



## Plateformes Git

Run et intégration au sein de Github & GitLab

- ◆ En direct sur le runner
- ◆ Dispatch distant
- ◆ Dispatch local
- ◆ Support des variantes de runner :
  - Linux / Ubuntu
  - MacOS
  - Windows

**Github**



Github Action fournie

```
jobs:
 melodium:
 uses: melodium-tech/github-actions/.github/workflows/melodium.yml@v0.9
 with:
 command: run .melodium-ci/Compo.toml --api_token "\<YOUR_JOB_API_TOKEN>" --output_direct
 artifact-path: 'logs/'
 secrets:
 token: ${ secrets.token }
```

**GitLab**



GitLab Template Workflow

```
include:
 - component: gitlab.com/melodium/gitlab-ci/melodium@v0.9
 inputs:
 job_name: melodium-ci
 command: run .melodium-ci/Compo.toml --api_token "\<YOUR_JOB_API_TOKEN>" --output_direct

melodium-ci:
 artifacts:
 paths:
 - logs
```

# Intégration



## Traitements de reporting

setServiceState

- Remontée des status et checks auprès des API Github & GitLab



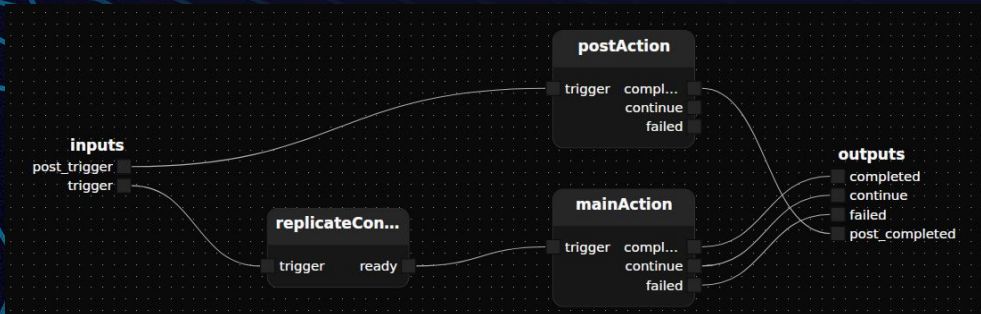
# Intégration



## Github

Outil de migration disponible :

- Transcrit la CI GHA d'un projet en Mélodium
- Permet de convertir des Github Actions tierces en traitements Mélodium



.YAML → .MEL

<https://cadence.ci>

Github Checkout action :

```
10 #[github_action(actions/checkout@v5)]
11 #[generated(true)]
12 treatment checkout[contexts: JavaScriptEngine, logger: Logger](
13 var clean: string = "true",
14 var fetch_depth: string = "1",
15 var fetch_tags: string = "false",
16 var filter: string = "",
17 var github_server_url: string = "",
18 var lfs: string = "false",
19 var path: string = "",
20 var persist_credentials: string = "true",
21 var ref: string = "",
22 var repository: string = "${{ github.repository }}",
23 var set_safe_directory: string = "true",
24 var show_progress: string = "true",
25 var sparse_checkout: string = "",
26 var sparse_checkout_cone_mode: string = "true",
27 var ssh_key: string = "",
28 var ssh_known_hosts: string = "",
29 var ssh_strict: string = "true",
30 var ssh_user: string = "git",
31 var submodules: string = "false",
32 var token: string = "${{ github.token }}"
33)
34 input post_trigger: Block<void>
35 input trigger: Block<void>
36 output completed: Block<void>
37 output continue: Block<void>
38 output failed: Block<void>
39 output post_completed: Block<void>
40 model node contexts: JavaScriptEngine()
```

# Intégration



## Github

Traitement `runAction` :

- Reproduit le comportement d'une Github Action
- Supporte la syntaxe Github

Traitements `githubStringEval`, `postGithubState...`

<https://cadence.ci/en/docs/from-github>

Actions Github de type :

- JavaScript / Node
- Docker
- Composites

```
100 postAction: runAction[contexts = node_contexts, logger = logger](
101 commands = "node .melodium-ci/github/actions/third/swatinem/rust_cache/ref_v2/dist/save/index.js",
102 if = "success() || env.CACHE_ON_FAILURE == 'true'",
103 name = "post:rustCache"
104)
```

# Intégration



## Infrastructure

Intégration générique dans l'infrastructure, sur toute machine.

- Bare install
- Conteneurs
- Intégration Kubernetes

<https://cadence.ci/en/docs/clusters/self-managed>

Makes engine available for distribution

**Usage:** `melodium dist [OPTIONS] --port <PORT> --recv-key <RECV_KEY> --send-key <SEND_KEY>`

### Options:

|                                                    |                                                                 |
|----------------------------------------------------|-----------------------------------------------------------------|
| <code>-i, --ip &lt;IP&gt;</code>                   | IP to listen on                                                 |
| <code>-p, --port &lt;PORT&gt;</code>               | Port to listen on                                               |
| <code>-c, --certificate &lt;CERTIFICATE&gt;</code> | Certificate chain to use for TLS encryption (PEM format)        |
| <code>-k, --key &lt;KEY&gt;</code>                 | Key to use for TLS encryption (PKCS8 PEM format)                |
| <code>-r, --recv-key &lt;RECV_KEY&gt;</code>       | Key expected to authenticate remote engine                      |
| <code>-s, --send-key &lt;SEND_KEY&gt;</code>       | Key to authenticate with remote engine                          |
| <code>--localhost</code>                           | Listen localhost (if ip is not set), using embedded certificate |
| <code>--disable-tls</code>                         | Disable TLS encryption                                          |
| <code>--wait &lt;WAIT&gt;</code>                   | Time (in seconds) to wait for a distant engine to connect       |
| <code>--duration &lt;DURATION&gt;</code>           | Maximal duration (in seconds) for work to be made               |

| Platform                                | Notes                                   |
|-----------------------------------------|-----------------------------------------|
| <code>aarch64-apple-darwin</code>       | ARM64 macOS (11.0+, Big Sur+)           |
| <code>aarch64-pc-windows-msvc</code>    | ARM64 Windows MSVC                      |
| <code>aarch64-unknown-linux-gnu</code>  | ARM64 Linux (kernel 4.1, glibc 2.17+)   |
| <code>aarch64-unknown-linux-musl</code> | ARM64 Linux with MUSL                   |
| <code>i686-pc-windows-gnu</code>        | 32-bit MinGW (Windows 7+)               |
| <code>i686-pc-windows-msvc</code>       | 32-bit MSVC (Windows 7+)                |
| <code>i686-unknown-linux-gnu</code>     | 32-bit Linux (kernel 3.2+, glibc 2.17+) |
| <code>i686-unknown-linux-musl</code>    | 32-bit Linux with MUSL                  |
| <code>x86_64-apple-darwin</code>        | 64-bit macOS (10.12+, Sierra+)          |
| <code>x86_64-pc-windows-gnu</code>      | 64-bit MinGW (Windows 7+)               |
| <code>x86_64-pc-windows-msvc</code>     | 64-bit MSVC (Windows 7+)                |
| <code>x86_64-unknown-linux-gnu</code>   | 64-bit Linux (kernel 3.2+, glibc 2.17+) |
| <code>x86_64-unknown-linux-musl</code>  | 64-bit Linux with MUSL                  |

+ S390x, RISC-V, Wasm, etc.



**Suite**

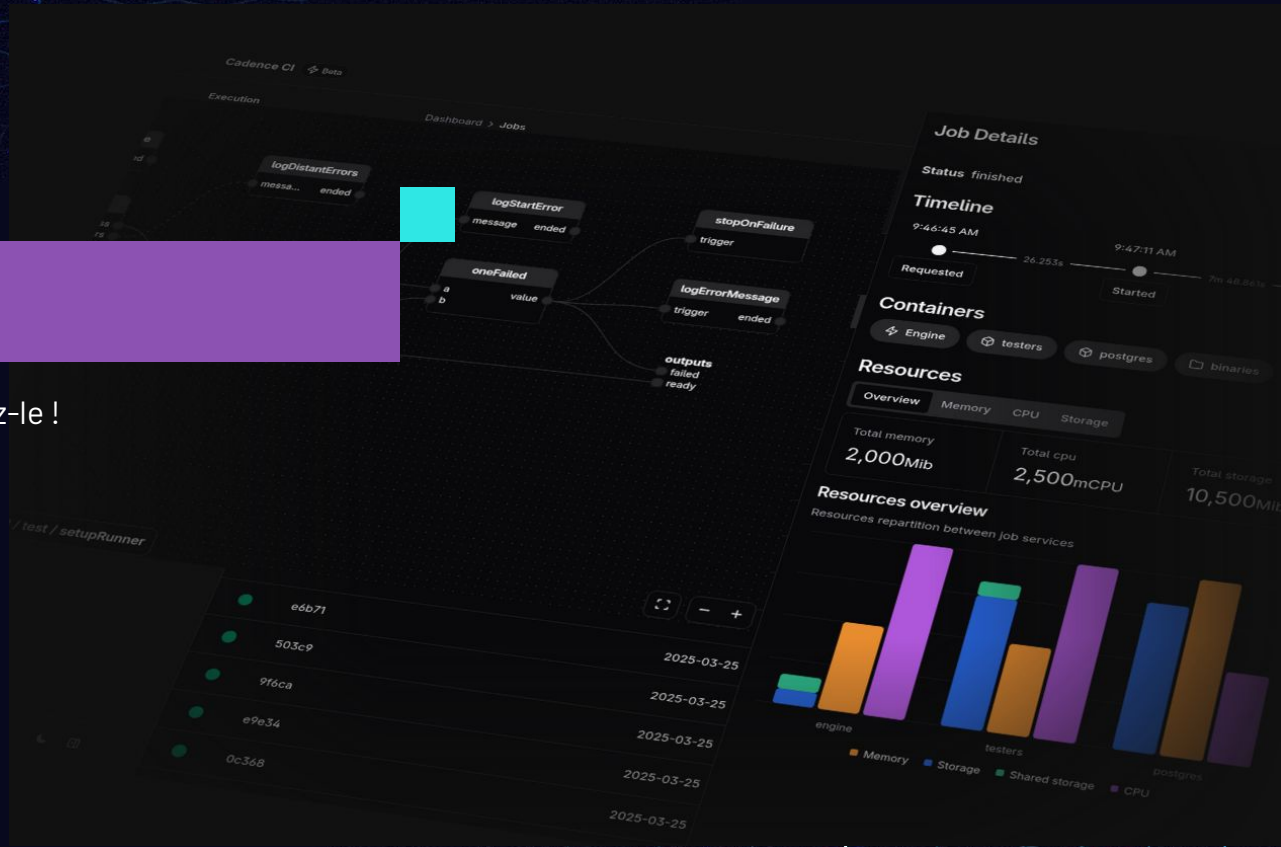
**05**

# Cadence.CI

Lancé fin 2025 en bêta, essayez-le !

- Amélioration des perfs
- Réduction des coûts

→ <https://cadence.ci>



Compositeur Studio

Fichiers Paramètres Aide

Project  
ci

tree  
on\_image  
buildTestAya  
buildTestAyaForArch

```
33 }
34 input trigger: Blockvoid->
35 output finished: Blockvoid->
36 model runner: CIGRunnerEngine()
37 {
38 build: stepOn[Logger = logger, runner = runner]({
39 commands = [raw Commands({
40 {bash -c "set -o allexport && source /tmp/github.env && set +o allexport
41 cargo hack build --all-targets --feature=powerset --exclude aya-ebpf --exclude aya-ebpf-bindings --exclude aya-log-ebpf --exclude integration-ebpf --ex
42 }},
43 {bash -c "set -o allexport && source /tmp/github.env && set +o allexport
44 cargo hack test --all-targets --feature=powerset --exclude aya-ebpf --exclude aya-ebpf-bindings --exclude aya-log-ebpf --exclude integration-ebpf --ex
45 }},
46 {bash -c "set -o allexport && source /tmp/github.env && set +o allexport
47 cargo hack test --doc --feature=powerset --exclude aya-ebpf --exclude aya-ebpf-bindings --exclude aya-log-ebpf --exclude integration-ebpf --exclude int
48 }},
49]},
50 environment = |wrap-Environment>({
51 environment({
52 has({
53 |entry({
54 "RUST_BACKTRACE",
55 "EXIT"
56 }},
57 |entry({
```

# Compositeur.Studi

Version alpha bientôt disponible,  
ajoutez-vous !

→ <https://compositeur.studio>



# Plus encore ?

Plein d'évolutions en cours et à faire, et de possibilités !

Ouvert à contributions et cas d'usages !



# Merci !

## Des questions ?

On est au Palace à Nantes, venez prendre un café ☕

Ou meet : <https://cal.com/qvignaud>

quentin.vignaud ✉ [melodium.tech](mailto:quentin.vignaud@melodium.tech)

<https://melodium.tech>

**CREDITS:** This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)