# Nouveautés Trackbone

## Version 3.2


- Jérémie Lesage

- 30/08/2024

# Plan

1. Config Default (v3.1)
2. Petites Corrections
3. Amélioration des logs
4. Fork Cue Flow
5. Définir les dépendances des taches
6. run.finally

# Config Default (v3.1)

- Depuis la version 3.0, 2 nouveaux paramètres existent
  - `--add-services`
  - `--add-children`
- Pour concourse on a défini fichier `ci/config-default.json`
- Pris en compte uniquement si `-t in_ci`

# ci/config-default.json

```
{
  "add-services": [
    "alertmanager",
    "blackbox-exporter-core",
    "grafana",
    "prometheus-rules",
    "kube-prometheus-stack",
    "rancher2-ngot-client"
  ],
  "add-children": []
}
```

# Petites Corrections

Régressions induites par trackbone 3.0 :

- `trackbone shell`
  - la target zone n’était pas pris en compte ce qui provoqué un chargement extrêmement long du shell
- `trackbone inputs, ouputs & graph`
  - problème de pointeur

- `trackbone outputs`
  - il est obligatoire de définir la `line` manuellement
- `trackbone make-diff`
  - Correction sur le passage de paramètre du make-diff

# Amélioration des logs

- Plus précis en cas d'erreur
- Moins verbeux par défaut (pour la CI).
- Nouveau paramètre `--trace` (en plus de `--debug`)
- Ajout de la durée sur toutes les tasks
  - (avec `--trace` ou si la durée d'exécution est > 1s)
- Suppression du "totalDuration" qui était bugué

# Fork Cue Flow

# C’est quoi Cue Flow

- Outil interne à Cuelang
- C’est un moteur de workflow basée sur un dag (Directed Acyclic Graph)
- https://github.com/cue-lang/cue/tree/master/tools/flow

# Pourquoi Cue flow c’est bien ?

- Description des taches directement en cue
- Les entrées-sorties des taches sont fusionnées dans le contexte cue
- Détermine automatiquement les dépendances entre les taches
  - Dépendances implicites
- Mise à jour automatique des dépendances lors de l’exécution

# Pourquoi c’est pas bien ?

- Ça utilise l’API interne de cue
  - On on ne peut pas surcharger le code
- Le contexte Cue n’est pas Thread-Safe
  - Donc l’enregistrement des sorties peut provoquer des crashes
- L’algo de calcul des dépendances est foireux
  - Explosion de la consommation mémoire
  - Boucle infini
  - Conflit dans la résolution du schéma cue

- Problème de montée en charge (Si +100 tasks)
- Aggravation des perfs et des anomalies avec cue 0.8+
  - car le moteur de cue est plus exigeant
- Ce n’est pas maintenu

# Fork and Cut 🍴

Ce qui a était fait :

- Copie du package `tools/flow` dans trackbone
- Suppression des dépendances à l'api `internal`
- Suppression du calcul des dépendances implicites
- Simplification du code (boucle de traitement)
- Fusion du contexte cue uniquement à la fin (perf)

# Conséquence du fork

## Dépendances Explicites

*Il faut déclarer toutes les dépendances de toutes les taches*

# Définir les dépendances des taches

# _need

- On ajoute **une** ou **des** champ(s) commençant par _need.
- Ces champs doivent référencer une tache ou une **liste de tache** en fournissant son chemin complet (path).

# Exemple :

```
run: pre_tasks: {
    read_secret: vault.#Read & {
        path: "secret/my-secret"
        data: foo: string
    }
    write_file: file.#Write & {
        _need_read_secret: "run.pre_tasks.read_secret"
        name: "secret"
        content: read_secret.data.foo
    }
}
```

# Exemple :

```
run: pre_tasks: {
    write_file: file.#Write & {
        _need: ["run.pre_tasks.task1", "run.pre_tasks.task2"]
        _need_read_secret: "run.pre_tasks.read_secret"
        if boostrap {
            _need_boot: "run.pre_tasks.boot"
        }
        name: "secret"
        content: read_secret.data.foo
    }
    writes_values: _need_wf: "run.pre_tasks.write_file"
}
```

# Exemple Helm :

```
run: pre_tasks: {
    ma_tache: {
        ...
    }
    writes_values: _need: "run.pre_tasks.ma_tache"
}

helm: {
    valuies: {
        ma_conf: run.pre_tasks.ma_tache.value
    }
}
```

# Exemple Terraform :

```
run: pre_tasks: {
    ma_tache: {
        ...
    }
    write_tfvars: _need: "run.pre_tasks.ma_tache"
}

tfvars: {
    ma_conf: run.pre_tasks.ma_tache.value
}
```

# Exemple Ansible :

```
run: pre_tasks: {
    ma_tache: {
        ...
    }
    write_extra_vars: _need: "run.pre_tasks.ma_tache"
    write_inventory: _need: "run.pre_tasks.ma_tache"
}

ansible: {
    extra_vars: {
        ma_conf: run.pre_tasks.ma_tache.value
    }
    inventory: {
        ...
    }
```

# Exemple Env :

```
run: pre_tasks: {
    ma_tache: {
        ...
    }
    write_environ: _need: "run.pre_tasks.ma_tache"
    write_environ_podman: _need: "run.pre_tasks.ma_tache"
}
env_vars: {
    ma_conf: run.pre_tasks.ma_tache.value
}
```

# Dépendances `pre_tasks` - `post_tasks`

- Les “post_tasks” sont toujours exécutées après les “pre_tasks”
- Aucun intérêt de faire des dépendances entre elles

# Plus de `_vault_login`

On a remplacé tous les

```
_vault_login: xxx
url: _vault_login.url
```

par

```
_need_vault: "run.pre_tasks.login"
url: run.pre_tasks.login.url
```

# Dépendances dans les `#TaskGroup`

- Les sous-taches dans un `#TaskGroup` peuvent se référencer entre elles.
- Mais une sous-taches ne peut pas référencer une tache externe.
- `tasks.nom_subtask`

# Exemple :

```
run: pre_tasks: mon_tunnel: {
    tb.#TaskGroup
    _need_vault: "run.pre_tasks.vault_mazone"
    tasks: {
        port: random.#AvailablePort
        make_tunnel: exec.#Run & {
            _need: "tasks.port"
            cmd: "bastion-tunnel"
        }
        // Make sure postgres is reachable through the tunnel
        // before we run the action.
        check_tunnel: exec.#Run & {
            _need: ["tasks.make_tunnel", "tasks.port"]
            cmd: ["nix-shell", "--quiet", "--command", "sleep
        }
```

# Run Finally

# Rappel 🧗

Ordres d’exécution des taches

1. pre_tasks
2. actions
3. post_tasks

Si une tache échoue, les suivantes ne sont pas exécutées

# Évolution

On définit la notion de finally

1. pre_tasks
2. actions
3. post_tasks
4. finally

Les taches `finally` sont toujours exécutées, même en cas d’erreurs.

# Application kill_tunnel

```
run: finally: kill_tunnel: #Run & {
  cmd: ["k8s-access", "kill", config_id]
}
```

- Le `kill_tunnel` est toujours exécuté à la fin
  - donc après toutes les taches `post_tasks`
  - même en cas de problème

Ceci résous un problème de tunnel persistant sur nos postes.

# Questions ?