コンテナ仮想化技術とワークフロー言語によってデータ解析環境の可搬性と再現性を向上する

大田達郎
ライフサイエンス統合データベースセンター
2019/04/22 @ かずさDNA研究所

Agenda

- 自己紹介
- 解析環境の可搬性とは
- 可搬性の低下が引き起こす問題
- 可搬性の高い解析環境が可能にすること
- 可搬性を高めるために：コンテナとワークフロー言語
- 実例: RNA-seq pipeline comparison
- まとめ

自己紹介

- 大田達郎, 特任研究員 @ DBCLS
  - twitter, github: @inutano
- 遺伝学研究所 (静岡県三島市) 勤務
  - DDBJ のお隣さん
- 最近のお仕事
  - RDFによるNGSデータやサンプル情報の統合
  - コンテナ、ワークフロー言語を用いた解析パイプライン構築
  - 技術者交流会 (Pitagora Meetup) の開催
    - 関連: Workflow Meetup

解析環境の可搬性とは

🤔 新規の計算機上で既存の解析手順を実行するのにかかるコストは？

- 誰でも新しい計算機で即座に計算が回せる = 可搬性が高い
- 計算を回すまでのセットアップに時間がかかる = 可搬性が低い
- 特定の計算機でしか動かない = 可搬でない

可搬性が低いと起きる問題

- 特定の計算機に計算が集中し、データの増加に対応できない
  - ジョブ投入に時間がかかる
  - ディスクが埋まる
  - オペレーションが特定の人員に集中する
- 解析結果に対する外部評価ができない
  - 外部での検証ができない = 解析手順の信頼性が低下
- エラー時の原因追求が困難
  - 環境構築が原因のエラーは解決に時間がかかる

可搬性が高いと可能になること

- 新しい計算機が手軽に使えるようになる
  - 計算機資源がスケールする
  - オンデマンドな計算機資源の調達も可能
    - クラウドとの親和性
- 解析環境の共有、配布が容易
  - 第三者による再実行による検証が可能
- 再現性の向上
  - **Nヶ月前の解析がちゃんと動く**

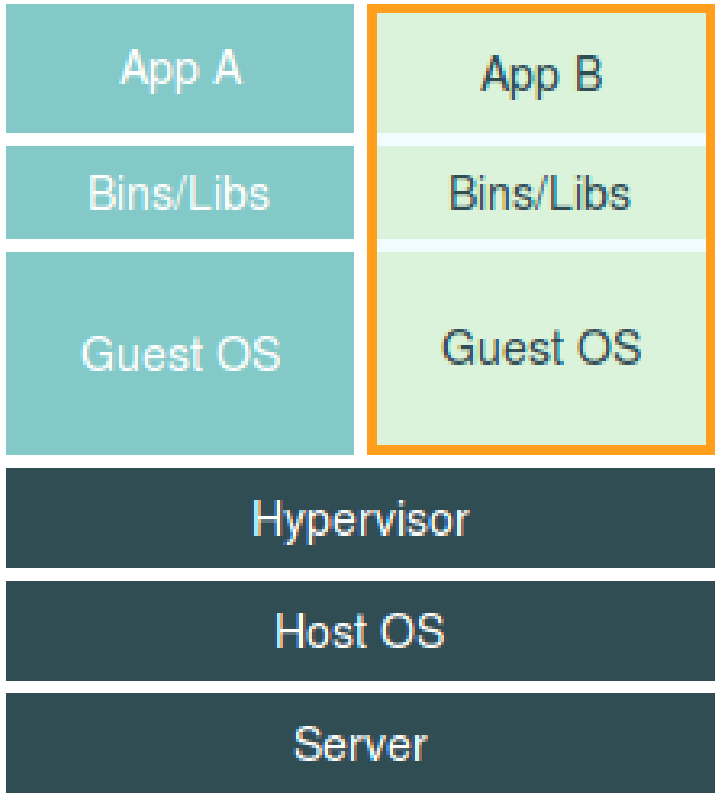
可搬性を高めるために

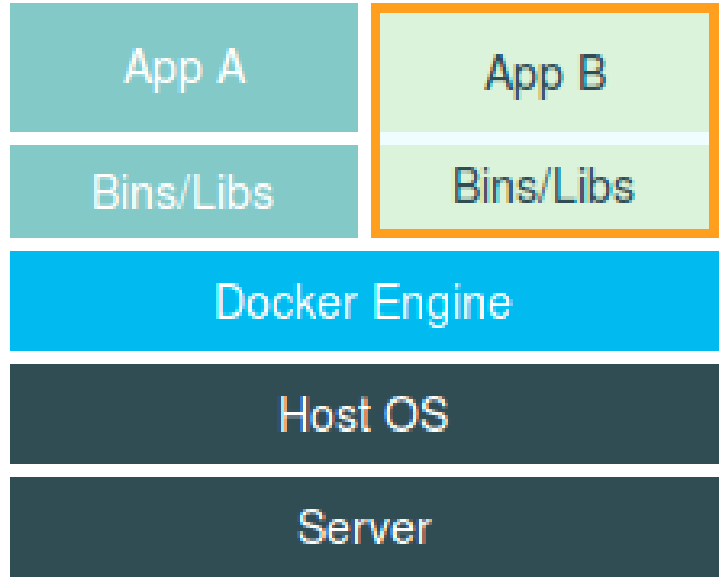
「コンテナ」と「ワークフロー言語」

- **コマンドラインツール**（ツール）をコンテナでパッケージング
  - スクリプトの依存関係
  - バイナリのビルド手順
  - ソフトウェアのバージョン
- **実行手順** をワークフロー言語でパッケージング
  - 実行時パラメータ
  - ツールの組み合わせ (ワークフロー)

コンテナによるパッケージング

コンテナ仮想化





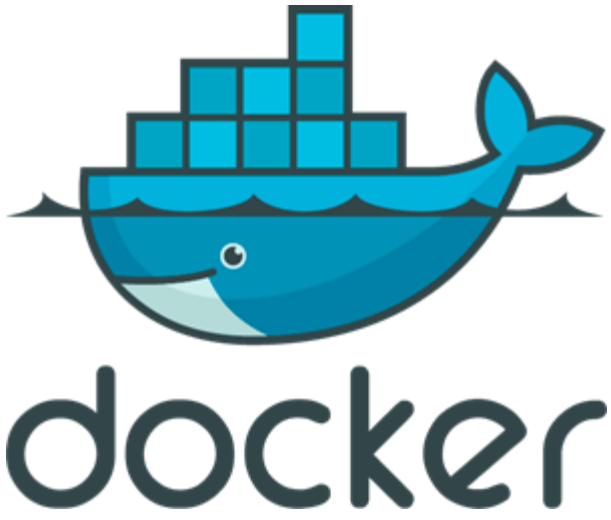
Virtual Machine vs Docker

https://blog.jayway.com/2015/03/21/a-not-very-short-introduction-to-docker/

各種コンテナエンジン

- Docker
- Singularity
- その他のコンテナエンジン
  - uDocker, shifter, rocket, podman, etc.

Docker



- 最も有名で普及しているコンテナエンジン
  - モダンな Web application 開発では避けて通れない
- root 権限で起動する Docker daemon を通じてコントロール
  - 共用マシンではセキュリティリスクに
  - 個人用マシンであれば Docker で何ら問題ない

https://www.docker.com

Singularity

- コンテナの実行に root 権限が不要
  - Docker の root 権限問題が発生しない
- Docker イメージも利用可能
- GridEngine などのジョブスケジューラとの親和性

https://www.sylabs.io/singularity/

コンテナを起動してコマンドを実行する

Docker

```
$ ls
SRR000001.fastq
$ docker run -it --rm \
    -v $(pwd):/work \
    -w /work \
    biocontainers/fastqc:v0.11.5_cv3 \
    fastqc \
    SRR000001.fastq
```

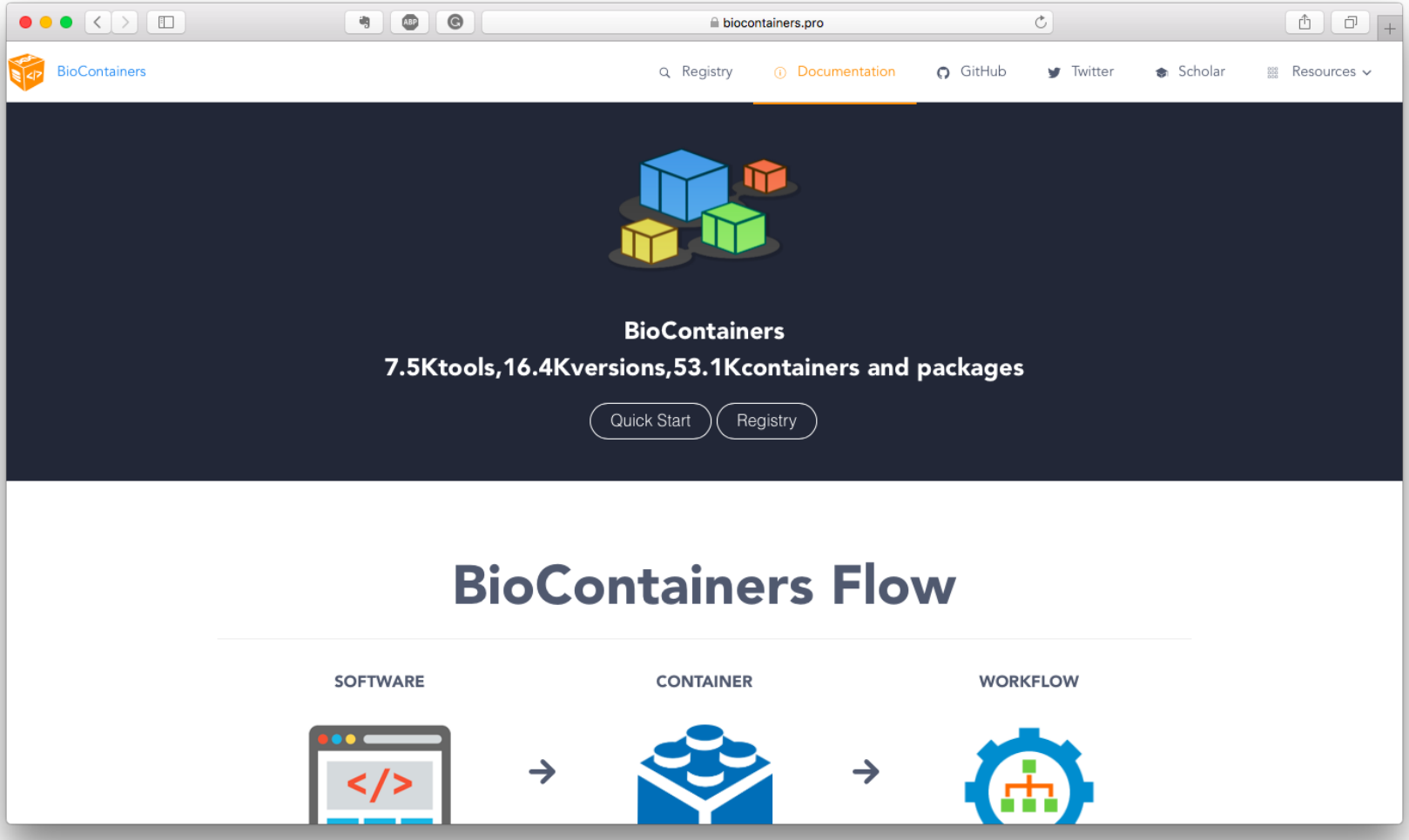
Singularity

```
$ singularity pull docker://biocontainers/fastqc:v0.11.5_
$ ls
fastqc-v0.11.5_cv3.simg
$ singularity exec fastqc-v0.11.5_cv3.simg \
    fastqc SRR000001.fastq
```

コンテナ仮想化をサポートするプロジェクト

- 起動補助
  - Docker compose
- コンテナのためのリソースマネジメント = オーケストレーション
  - k8s
  - GridEngine
- BioContainers
  - bioconda レシピを使ってコンテナイメージをビルドする

BioContainers



https://biocontainers.pro

BioContainers

- bioconda: bioinformatics ツールのインストールを簡易化するプロジェクト
  - ツールインストールの「レシピ」を収集
- bioconda レシピを流用してコンテナイメージを作成、配布
- コンテナイメージを自分でビルドしなくて便利だが探すのが大変
  - Bioconda
  - BioContainers Registry
  - Docker Hub / BioContainers
  - Quay.IO / BioContainers
- Bioconda の検索から辿るのがおすすめ

コンテナがないときは？

作りましょう。たとえばこんな感じです:

- スクリプトないしバイナリを GitHub に置く
- GitHub からソースを取得してビルドする Dockerfile を書く
- Dockerfile を GitHub でホスト
- Quay.IO で GitHub レポジトリ連携の自動ビルドをセット
- GitHub でリリースを打つとコンテナが勝手にビルドされる
- pull して使う

コンテナ作成のベストプラクティス

- Docker container best practice
- "1ツール1コンテナ" を推奨
  - ツール単位でのバージョン管理や変更を加えるのが容易に
- コンテナサイズが小さくなるように
  - 色々のテクニックがあります
  - 100MBくらいには抑えたい
- コンテナの再利用を意識する
  - 同じベースイメージを使い回す
- 公開できないツールはどうするか
  - 本当に表に出せませんか？
    - publicな便利ツールを使えない = 自前でやるのは大変
  - そもそもコンテナ化する必要ありますか？

ワークフロー言語によるパッケージング

ワークフロー言語

コンテナ仮想化のおかげでツールの移植が簡単になる
=> ワークフローを共有するためのフレームワークの開発も活発に

- DSL型
  - プログラミング言語のように書ける柔軟性
  - 例: nextflow, WDL, snakemake
- データ型
  - 共有、配布を主目的とし、パースや生成が (比較的) 楽
  - 例: CWL, Galaxy workflow XML

DSL型 vs データ型

- DSL型
  - pros
    - スクリプト言語で書かれた既存のWFを移植しやすい
  - cons
    - 編集するにはそれぞれの文法を覚える必要がある
    - エンジンが固定
- データ型
  - pros
    - パーサやエンジンが揃っているので配布しやすい
  - cons
    - 複雑な処理は書き下せない場合も (スクリプトに書いてそれを実行するしかない)

=> 身内で完結するならDSL、不特定多数に配るならデータ型

Common Workflow Language (CWL)

- BOSC生まれ、GitHub育ちの完全OSSプロジェクト
- 規約 (specification) と標準実装 (reference implementation)
- YAMLベースで input/command/output を記述する
- Bioinfo に限らない, 物理学分野などでも使っている人がいる
- 日本にも5人くらいのコミッタがいる

CWL で何ができるか

- Command Line Tool のパッケージング
- Workflow のパッケージング
- 可視化
- エディタのサポート
- 複数のワークフローエンジンで実行が可能
  - cwltool, arvados, toil, CWL-airflow, REANA, Cromwell, CWLEXEC
  - その他 Galaxy, Taberna などでもサポートのための開発中

Command Line Tool の定義ファイル `fastq-dump.cwl`

```
cwlVersion: v1.0
class: CommandLineTool
hints:
  DockerRequirement:
    dockerPull: quay.io/inutano/sra-toolkit:v2.9.0

baseCommand: [fastq-dump]
inputs:
  sraFiles:
    type: File[]
    inputBinding:
      position: 1
  split_files:
    type: boolean?
    default: true
    inputBinding:
      prefix: --split-files
outputs:
  fastqFiles:
    type: File[]
    outputBinding:
      glob: "*fastq*"
```

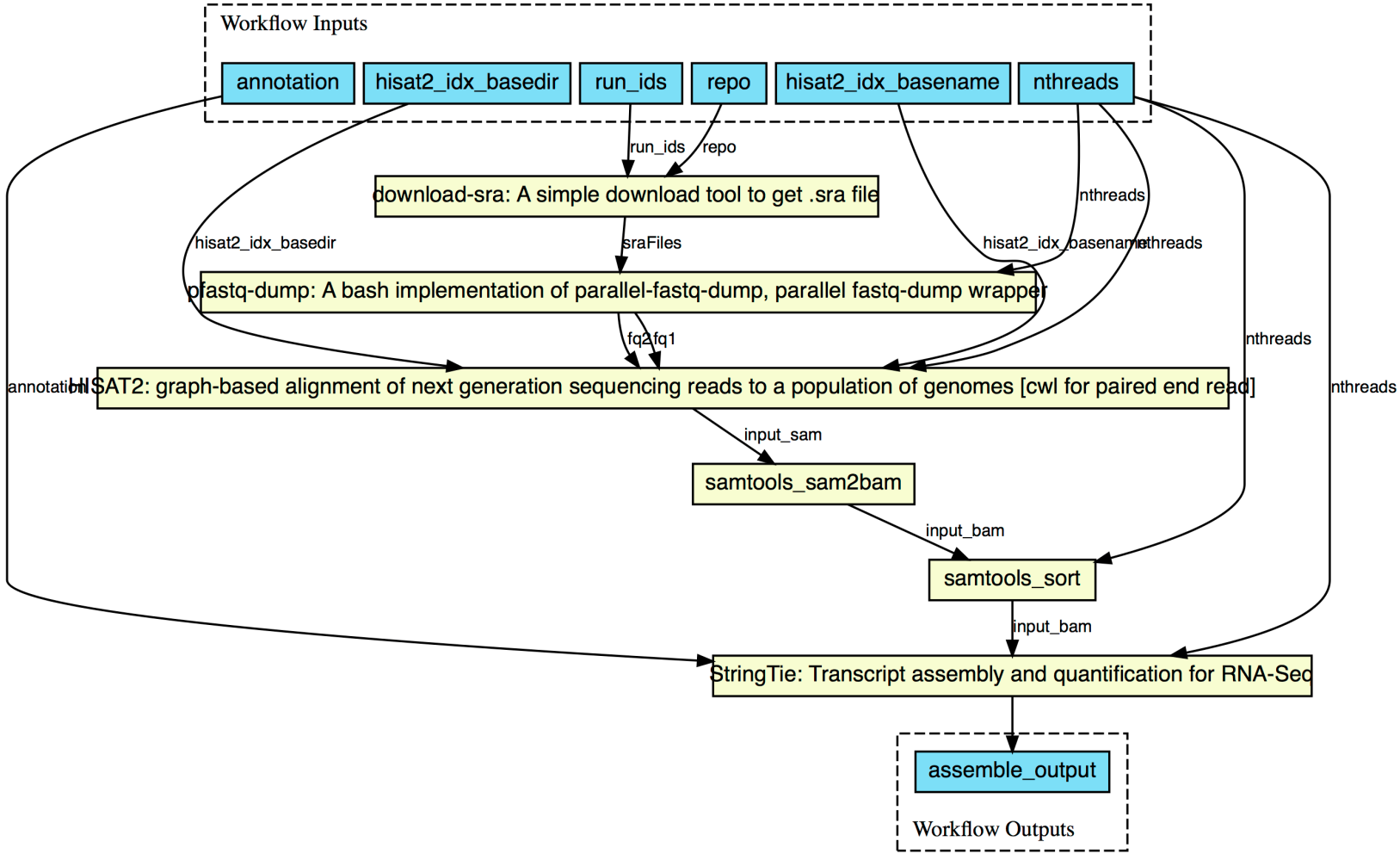
Workflow の定義ファイル `fastqc_wf.cwl`

```
cwlVersion: v1.0
class: Workflow

inputs:
  sra_files: File[]
outputs:
  fastqc_result:
    type: File[]
    outputSource: fastqc/fastqc_result
steps:
  pfastq_dump:
    run: pfastq-dump.cwl
    in:
      sraFiles: sra_files
    out:
      [fastqFiles]
  fastqc:
    run: fastqc.cwl
    in:
      seqfile: pfastq_dump/fastqFiles
    out:
      [fastqc_result]
```

view.commonwl.org

GitHub に置いてある CWL ファイルをレンダリング

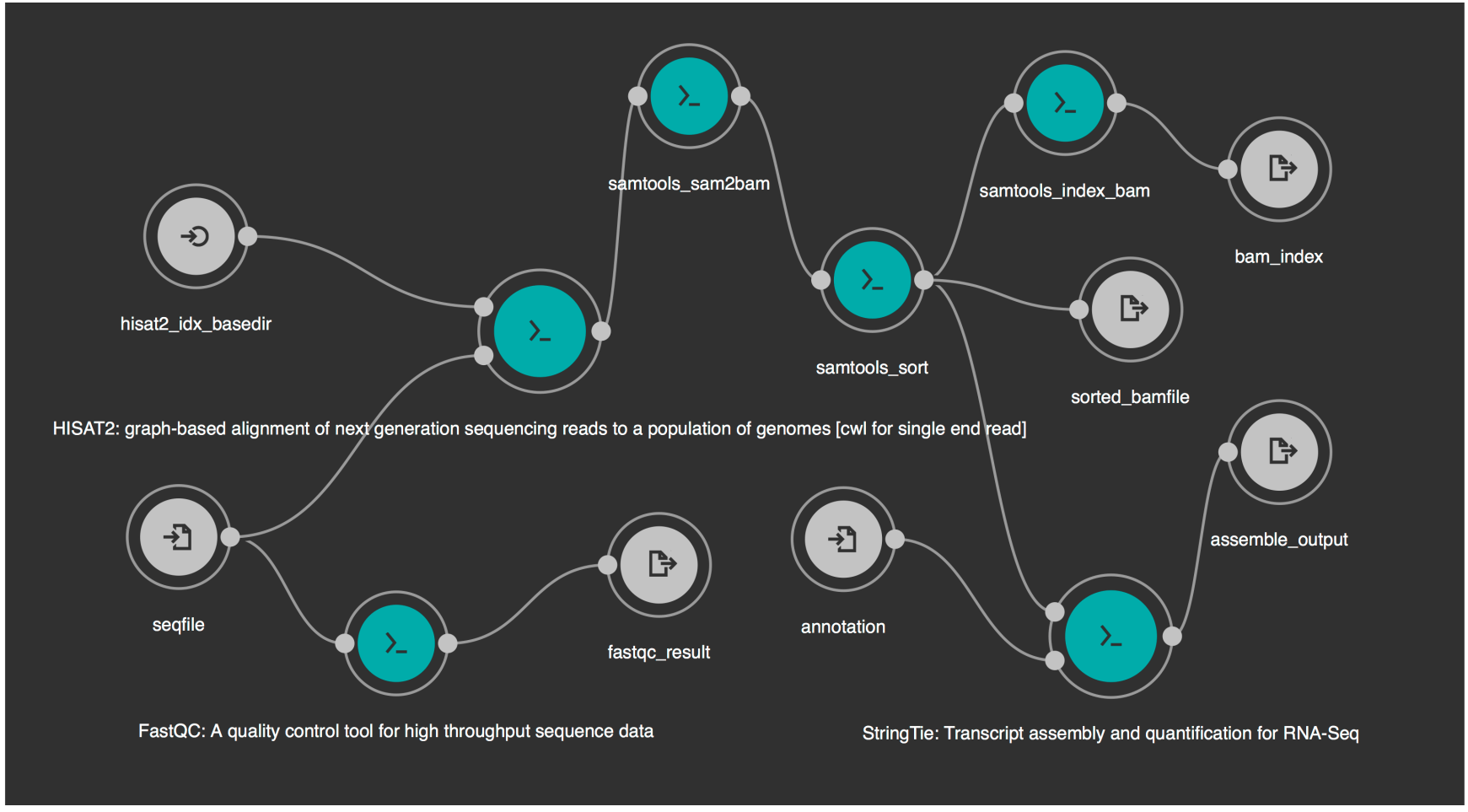


エディタサポート

- Rabix Composer
- Atom
- Vim
- Emacs
- VScode
- IntelliJ
- gedit
- Sublime Text

Rabix Composer

GUI でのCWLの編集と可視化をサポート



標準実装 cwltool で CWL workflow を実行する

```
$ cwltool fastqc_wf.cwl --sra_files SRR000001.sra
---
$ cat job_conf.yml
sra_files:
  - SRR000001.sra
  - SRR000002.sra
  - SRR000003.sra
$ cwltool fastqc_wf.cwl job_conf.yml
```

https://github.com/common-workflow-language/cwltool/

Implementations

Name	Desc	Platform
cwltool	Reference implementation of CWL	Linux, OS X, Windows, local execution only
Arvados	Distributed computing platform for data analysis on massive data sets	AWS, GCP, Azure, Slurm
Toil	A workflow engine entirely written in Python	AWS, Azure, GCP, Grid Engine, HTCondor, LSF, Mesos, OpenStack, Slurm, PBS/Torque
CWL-Airflow	Package to run CWL workflows in Apache-Airflow	Linux, OS X
REANA	RE usable ANAlyses	Kubernetes, CERN OpenStack (OpenStack Magnum)
Cromwell	Cromwell workflow engine	Google, HTCondor, Local, LSF, PBS/Torque, SGE, Slurm, TES
CWLEXEC	Apache 2.0 licensed CWL executor for IBM Spectrum LSF	IBM Spectrum LSF 10.1.0.3+

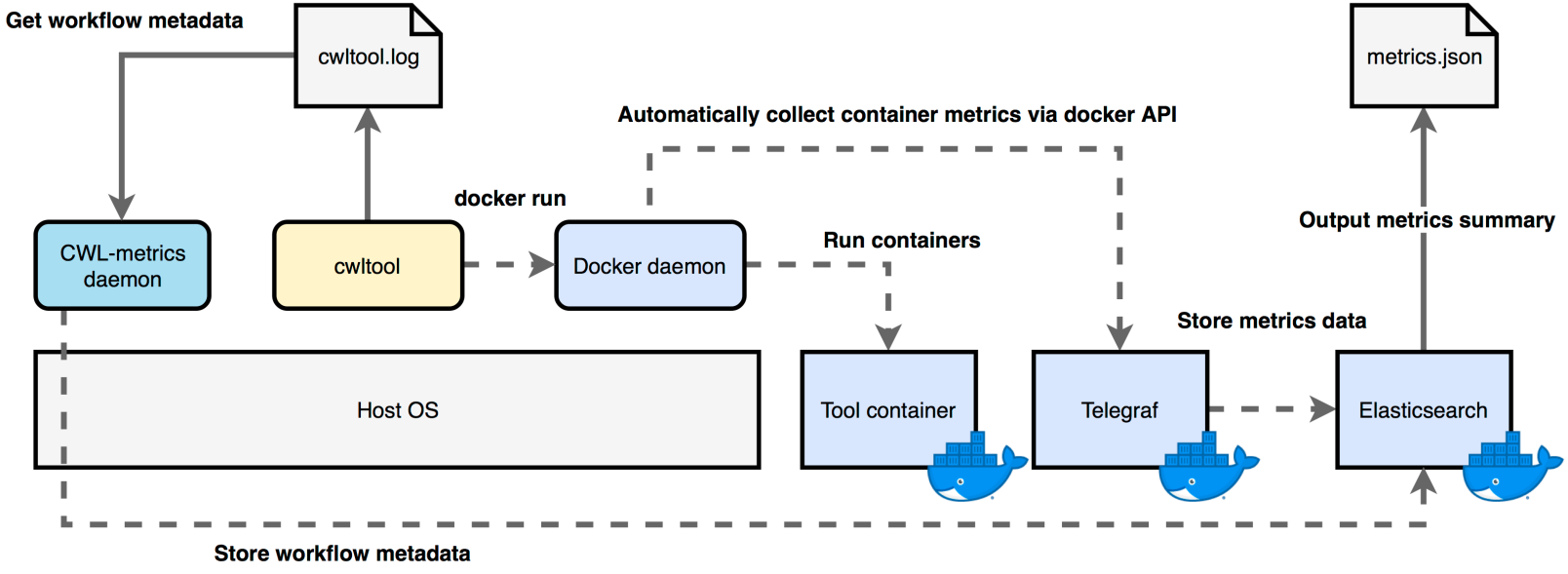
CWLでのパッケージングの実例

RNA-seq quantification パイプライン比較をCWLで

1. ワークフローを組む
2. BioContainers でツールのコンテナを探す
3. なければ自分でコンテナをパッケージングする
    3.1. GitHub に Dockerfile を push してリリースタグを振る
    3.2. Quay.io で自動ビルド
4. CWLを書く
5. 動かす、可視化

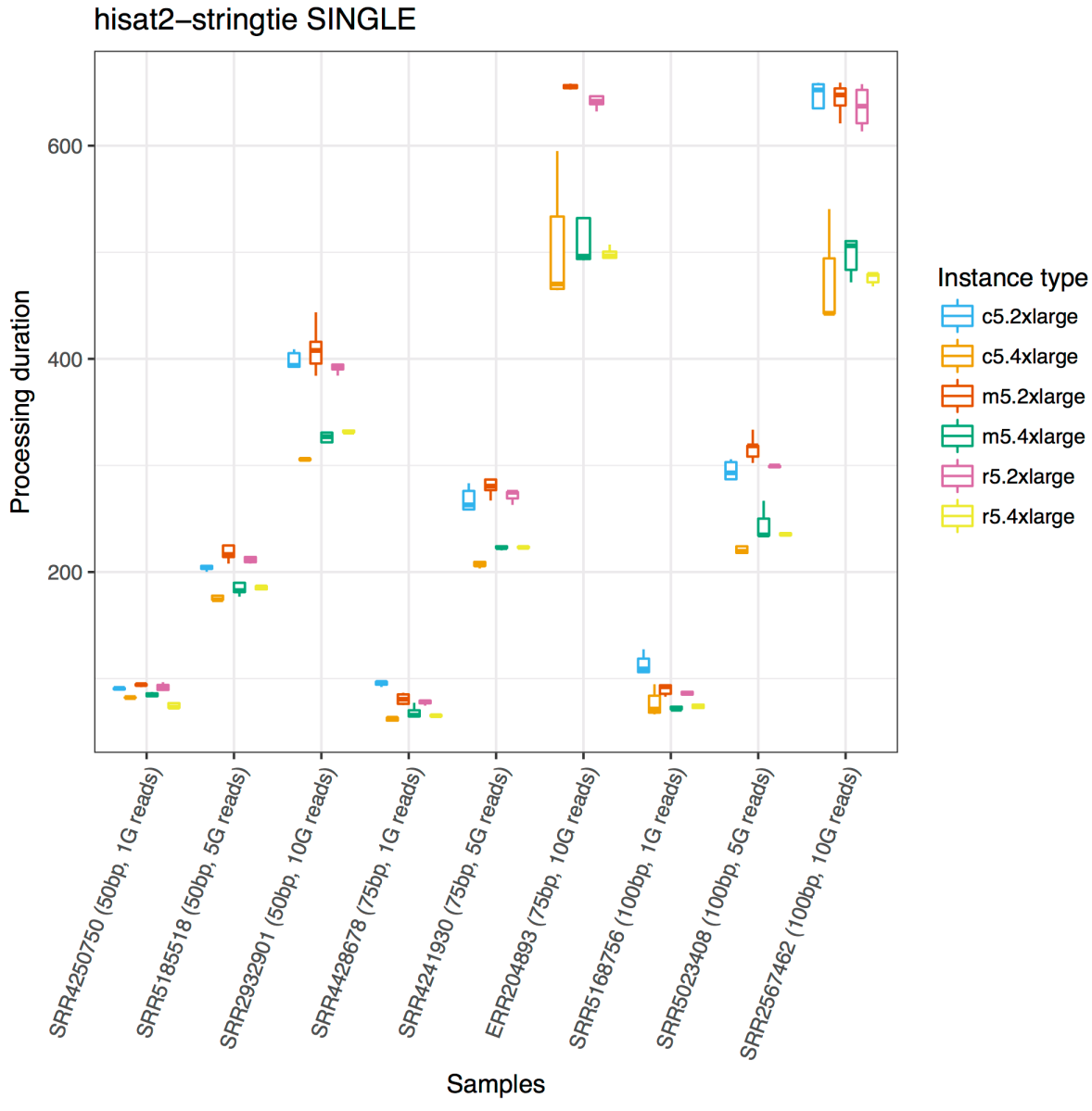
CWL-metrics

- CWL で記述されたワークフローの要求リソース量を計測
- コンテナごとのリソース消費とCWL記述を統合する

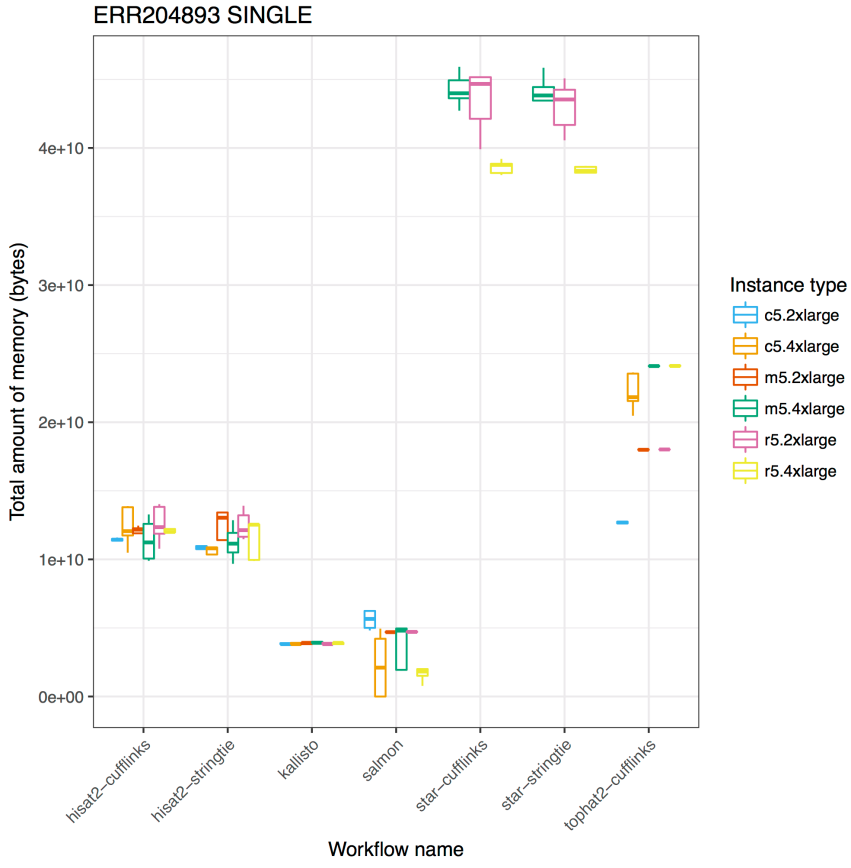


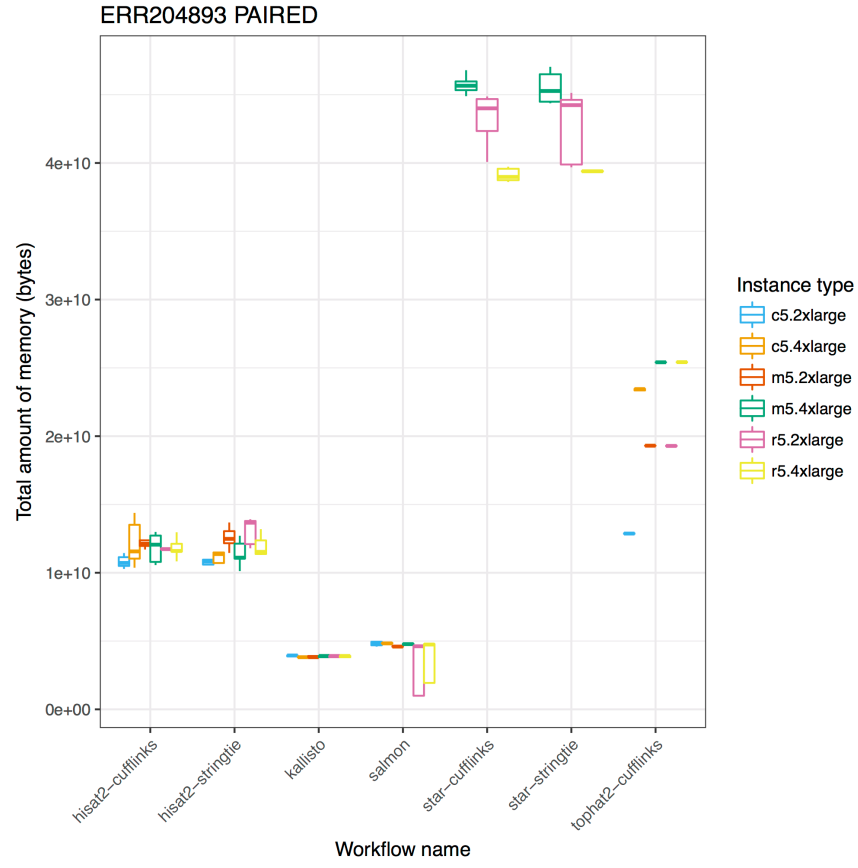
https://inutano.github.io/cwl-metrics/

HiSAT2-stringtie WF



Memory requirement





CWL workflow 実例

- 公開WF多数
  - GitHub
  - Dockstore
- 事例
  - EBI MGnify のCWL化
  - DDBJ スパコン上で稼働する pipeline システムを CWL ベースで開発
  - NCBI pgap が CWL でリリースされる

CWL で書かなくてはいけないのか？

慣れた言語で実装する方が速い: 時間と手間をかけるメリットはあるか？
以下のような項目を考慮すべき:

- そのワークフローを動かすのは誰？
  - 自分、グループメンバー、所外のコラボレータ、世界のどこかの誰か
- どれくらいメンテナンスしなくてはいけないのか？
  - 今動けばあとはどうでもいい or 数年は動いていてほしい
  - 自分以外の誰かにメンテを引き継ぎたい
- 今少し苦労するか、後で大変な思いをするか

おわりに

- コンテナとワークフロー言語を活用して解析環境の可搬性が向上
  - フレームワークは用途に合ったものを
- 「数ヶ月後に実行しても動く」安心感
  - レビュワーの無茶振りが怖くない
- 共有して他人に使ってもらって初めて気づく改善点が沢山ある
  - 共有した方が結果的にいいものに (人はやさしい)
  - 頑張りが無駄にならない