

Organizowanie potoków przetwarzania w bioinformatyce

Wprowadzenie do Nextflow z przykładami kodu

Piotr Suszyński

7 maja 2025

Plan wykładu

Wprowadzenie do systemów zarządzania przepływami pracy

Podstawowe koncepcje Nextflow

nf-test i nf-core

Przykład przepływu pracy DNA NGS

Podsumowanie

Wprowadzenie do systemów zarządzania przepływami pracy

Dlaczego systemy zarządzania przepływami pracy?

- Potoki przetwarzania w bioinformatyce obejmują wiele narzędzi o różnorodnym charakterze (natywne nierozproszone, rozproszone, skrypty Perl, R i Python).
- Wyzwania:
 - Zarządzanie środowiskami oprogramowania, różnorodnym sposobem instalacji i uruchamiania, zależnościami, wersjami narzędzi.
 - Zapewnienie powtarzalności na różnych platformach.
 - Skalowanie do dużych zbiorów danych, równoległe wykonywanie zadań na HPC lub w chmurze.
 - Złożoność grafu wywołań różnych narzędzi prowadzi do chaosu i błędów.

Dlaczego systemy zarządzania przepływami pracy?

- Systemy zarządzania przepływami pracy (Workflow Management Systems - WfMS) rozwiązują te problemy przez:
 - Automatyzację wykonywania zadań.
 - Deklaratywne definiowanie grafów przetwarzania, modularyzację i kompozycję.
 - Obsługę zależności i wsparcie dla konteneryzacji.
 - Ułatwienie zrównoleglenia wykonywania procesów.
 - Monitorowanie stanu wykonania, raportowanie a także wznawianie i reakcję na problemy.

Przykłady: Nextflow[1], Snakemake, Cromwell, Galaxy.

Porównanie Cromwell, Snakemake, Nextflow

Cecha	Cromwell	Snakemake	Nextflow
Język specyfikacji przepływu pracy	WDL	DSL + Python	DSL + Groovy
Dojrzałość i wsparcie przemysłowe	Wysoka, wspierane przez Broad Institute	Wysoka, ale wsparcie tylko środowiska akademickiego	Wysoka, wspierane przez Seqera Labs
Skalowalność	Wysoka – duże klastry, Terra	Średnia do wysokiej	Bardzo wysoka – chmura, hybrydowe środowiska, Seqera
Dynamiczne DAGi	Nie – statyczna struktura	Częściowo – ograniczone rozszerzenia	Tak – dynamiczne kanały i logika sterująca
Kod: czytelność i odporność na błędy	Duża – kod deklaracyjny i statycznie typowany, WDL to standard	Niestandardowy język specyfikacji reguł przeplatany z dynamicznie typowanym Pythonem	Groovy jest dynamicznie typowany i mniej popularny, dynamiczne transformacje kanałów mogą utrudniać zrozumienie logiki przetwarzania
Zarządzanie środowiskami	Docker, Singularity dostępne z dodatkową konfiguracją	Conda, Docker	Conda, Docker, Singularity
Wsparcie Kubernetes	Częściowe – backend, wymaga konfiguracji	Częściowe – prosty wrapper	Pełne – procesy jako pody, śledzenie cyklu życia, zarządzanie zasobami, autoskalowanie

- **Nextflow**: System WfMS dla skalowalnych, powtarzalnych przepływów pracy w bioinformatyce.
- Kluczowe cechy:
 - Język specyficzny dla domeny (DSL) oparty na Groovy.
 - Model programowania przepływu danych: Procesy uruchamiają się, gdy dane wejściowe są gotowe.
 - Wsparcie dla kontenerów (Docker, Singularity) i menedżerów pakietów (Conda).
 - Niezależność od platformy: lokalnie, HPC (np. SLURM) lub chmura (np. AWS).
 - Punkty kontrolne dla wznowiania pracy.

Dlaczego Nextflow? Silna społeczność (nf-core), przenośność i skalowalność. Popularne, dojrzałe rozwiązanie, stosowane produkcyjnie.

Podstawowe koncepcje Nextflow

Bloki konstrukcyjne Nextflow

- **Procesy (process):** Definiują zadania (np. uruchamianie bwa mem).
- **Kanały (channel):** Przekazują dane między procesami (kanały kolejkowe lub wartościowe).
- **Przepływy pracy (workflow):** Organizują procesy w graf przetwarzania.
- **Konfiguracja (config):** Dostosowują zasoby (np. CPU, pamięć) i profile.
- **Dyrektywy (directive):** Kontrolują wykonywanie procesów (np. `publishDir`).

Prosty przykład Nextflow

Prosty przykład: Zliczanie linii w pliku FASTQ.

```
1 #!/usr/bin/env nextflow
2
3 process NUM_LINES {
4     script:
5     """
6     zcat ${projectDir}/data/sample.fastq.gz | wc -l
7     """
8 }
9 workflow {
10     NUM_LINES()
11 }
```

Uruchamianie za pomocą: `nextflow run wc.nf`

Kanały i przepływ danych: Podstawy

- **Kanały:** Łączą procesy, przekazując dane (wejścia/wyjścia).
- **Typy kanałów:**
 - *Kanały kolejkowe:* Kolejka FIFO łącząca producenta i konsumenta (np. `Channel.fromPath("/data/*.fastq")`).
 - *Kanały wartościowe:* Pojedyncze wartości, mogą być konsumowane wielokrotnie (np. `Channel.value(42)` lub rezultat operatorów takich jak `first`, `collect` czy `sum`).
- **Transformacje:**
 - Modyfikacja: `map`, `flatMap`
 - Wybór: `filter`, `take`, `until`, `randomSample`.
 - Agregacja: `reduce`, `sum`, `max`
- **Równoległość:** Nextflow uruchamia zadania równolegle dla każdego elementu kanału, gdy dane wejściowe są gotowe.

Kanały: Łączenie, dzielenie, grupowanie

- **Łączenie i scalanie:**
 - `join`: Paruje elementy według klucza (np. ID próbki).
 - `mix`: Scala kanały bez dopasowania kluczy.
- **Dzielenie:**
 - `splitFastq`: Dzieli pliki FASTQ na fragmenty (np. by: 1000).
 - `branch`: Dzieli kanał na podstawie predykatu, umożliwiając warunkowe wykonanie.
- **Grupowanie:**
 - `groupTuple`: Grupuje według klucza (np. ID próbki, nazwa chromosomu).

Kanały: Przykład

```
1 params.vcfs = "${projectDir}/data/*.vcf"
2 params.metadata = "${projectDir}/data/metadata.csv"
3 include { SPLIT_VCF_BY_CHROM } from '../modules/local/split_vcf'
4 include { PLINK_MAKEBED } from '../modules/nf-core/plink2/makebed'
5
6 workflow {
7     vcf_ch = Channel.fromPath(params.vcfs)
8         .map { file -> [file.baseName, file] }
9
10    metadata_ch = Channel.fromPath(params.metadata)
11        .splitCsv(header: true)
12        .map { row -> [row.sample_id, row.group] }
13
14    chrom_ch = SPLIT_VCF_BY_CHROM(
15        vcf_ch.join(metadata_ch).map { id, vcf, group -> [id, vcf] }
16    )
17
18    grouped_ch = chrom_ch.map { id, chrom, vcf_data -> [chrom, vcf_data] }
19        .groupTuple()
20
21    branched_ch = grouped_ch.branch {
22        x: it[0] == 'chrX'
23        nonx: it[0] != 'chrX'
24    }
25
26    PLINK_MAKEBED(branched_ch.nonx)
27 }
```

Konfiguracja i skalowalność

- **Pliki konfiguracyjne (`nextflow.config`):**
 - Definiują zasoby, takie jak liczba rdzeni i dostępna pamięć.
 - Profile: `-profile docker` dla wyboru konteneryzacji.
 - Umożliwiają różne ustawienia dla procesów, różne dla testowania lub środowiska.
- **Skalowalność:**
 - Automatycznie równoległe wykonywanie zadań na podstawie kanałów wejściowych.
 - Wsparcie dla schedulerów HPC (SLURM, PBS), Kubernetes i platform chmurowych.
- Przykład: Konfiguracja konteneryzowanego FastQC z 2 CPU.

Konfiguracja i skalowalność - przykład

```
1 process FASTQC {
2   cpus 2
3   tag "${sample_id}"
4   conda "${moduleDir}/environment.yml"
5   container 'quay.io/biocontainers/fastqc:0.12.1--hdfd78af_0'
6
7   input:
8   tuple val(sample_id), path(reads)
9
10  output:
11  path("${sample_id}_fastqc.zip")
12
13  script:
14  """
15  fastqc -t ${task.cpus} ${reads} -o .
16  """
17 }
18
19 workflow {
20   reads_ch = Channel.fromFilePairs("${projectDir}/data/*_{1,2}.fastq.gz")
21   FASTQC(reads_ch)
22 }
```

Konfiguracja i skalowalność - environment.yml

- Możemy łatwo definiować środowiska Conda z potrzebnymi zależnościami.
- Wysokiej jakości moduły Nextflow definiują kilka opcji uruchamiania - użytkownik musi tylko wybrać profil.

```
1 name: my-env
2 channels:
3   - conda-forge
4   - bioconda
5   - defaults
6 dependencies:
7   - bioconda::fastqc=0.12.1
8   - bioconda::htslib=1.21
9   - bioconda::samtools=1.21
10  - bioconda::bcftools=1.21
11  - bwa=0.7.15
```


nf-test i nf-core

nf-test: Testowanie przepływów pracy

- **nf-test:** Framework do testowania przepływów pracy Nextflow.
- Zapewnia, że przepływy pracy generują oczekiwane wyniki.
- Kluczowe cechy:
 - Testy jednostkowe dla procesów i workflowów.
 - Automatyczne przechwytywanie i walidacja migawek.
 - Wsparcie dla testów zależnych od wyników innych procesów.
 - Może być nieco wolny.
- Przykład: Testowanie wykonania plink.

nf-test: Testowanie przepływów pracy - przykład

```
1 nextflow_process {
2   script "../main.nf"
3   process "PLINK2_MAKEBED"
4
5   test("small vcf") {
6     when {
7       process {""
8         input[0] = [
9           [ id:'test', single_end:false ],
10          file('test.bed'), file('test.bim'), file('test.fam')
11        ]
12        input[1] = '--hwe 1e-13 0.001'
13      ""}
14    }
15
16    then {
17      assertAll(
18        { assert process.success },
19        { assert snapshot(process.out).match() }
20      )
21    }
22  }
23 }
```

Uruchamianie: `nf-test run test_plink.nf`

nf-core: Przepływy pracy wspierane przez społeczność

- **nf-core[2]**: Projekt społecznościowy dla starannie przygotowanych, najlepszych praktyk przepływów pracy Nextflow.
- Cechy:
 - 130 przepływów pracy i ponad 1500 modułów.
 - Standaryzowane, dobrze udokumentowane i przetestowane.
 - Wykorzystuje Docker/Singularity dla powtarzalności.
 - Dostarcza narzędzie wiersza poleceń do instalacji modułów i tworzenia nowych modułów z szablonu.
- Narzędzia: `nf-core modules install`.
- Przykład: Uruchamianie `nf-core/rnaseq`.

```
nextflow run nf-core/rnaseq -input samplesheet.csv  
-genome GRCh37 -profile docker
```

Przykład przepływu pracy DNA NGS

Przepływ pracy DNA NGS: Variant calling

- Cel: Przetwarzanie danych DNA NGS w celu uzyskania wariantów w pliku vcf za pomocą bwa, samtools i GATK.
- Kroki:
 1. Dopasowanie do sekwencji referencyjnej za pomocą bwa mem.
 2. Sortowanie pliku BAM za pomocą samtools.
 3. Dodanie wymaganych przez GATK pól w headerze pliku BAM za pomocą samtools.
 4. Variant calling za pomocą GATK HaplotypeCaller.
- Przykład używa modułów nf-core a także subworkflowu wywołującego bwa i samtools.

Kod przepływu pracy DNA NGS

```
1 params.reads = "${projectDir}/data/*_{1,2}.fastq.gz"
2 params.outdir = "results"
3 params.bwa_index = "${projectDir}/ref/bwa_index.{amb,ann,bwt,pac,sa}"
4
5 include { FASTQ_ALIGN_BWA } from '../subworkflows/nf-core/fastq_align_bwa'
6 include { HAPLOTYPECALLER } from '../modules/nf-core/gatk4/haplotypecaller'
7
8 process ADD_READ_GROUP {
9     tag "${meta.id}"
10    publishDir "${params.outdir}/bam", mode: 'copy', overwrite: true
11    container 'biocontainers/samtools:1.21--h50ea8bc_0'
12
13    input:
14    tuple val(meta), path(bam), path(bai)
15    output:
16    tuple val(meta), path("${meta.id}.rg.bam"), path("${meta.id}.rg.bam.bai")
17
18    script:
19    """
20    samtools addreplacerg \\\
21    -r "ID:${meta.rgid}\\tSM:${meta.rgsm}\\tPL:${meta.rgpl}\\tLB:${meta.rglb}" \\\
22    -o ${meta.id}.rg.bam ${bam}
23    samtools index ${meta.id}.rg.bam
24    """
25 }
```

Kod przepływu pracy DNA NGS - kontynuacja

```
1 workflow {
2   reads_ch = Channel.fromFilePairs(params.reads)
3   .map { id, files -> [[id: id, single_end: false, rgid: "${id}_group",
4     rgsm: id, rgpl: 'ILLUMINA', rglb: 'lib1'], files] }
5   bwa_index_ch = Channel.fromPath(params.bwa_index)
6   .collect() // Grouping 5 index files into a single object
7   .map { index -> [[id: 'bwa_index'], index] }
8
9   // Triggering FASTQ_ALIGN_BWA subworkflow
10  FASTQ_ALIGN_BWA(reads_ch, bwa_index_ch, true, [], [])
11
12  bam_rg_ch = ADD_READ_GROUP(
13    FASTQ_ALIGN_BWA.out.bam.join(FASTQ_ALIGN_BWA.out.bai, by: 0)
14  )
15
16  // Preparing input channels for GATK
17  fasta_ch = Channel.fromPath(params.ref_fasta)
18  .map { fasta -> [[id: 'ref'], fasta] }
19  fai_ch = Channel.fromPath(params.ref_fasta_fai)
20  .map { fai -> [[id: 'ref'], fai] }
21  dict_ch = Channel.fromPath(params.ref_dict)
22  .map { dict -> [[id: 'ref'], dict] }
23
24  gatk_input_ch = bam_rg_ch.map { meta, bam, bai -> [meta, bam, bai, [], []] }
25
26  vcf_ch = HAPLOTYPECALLER(
27    gatk_input_ch, fasta_ch, fai_ch, dict_ch, [ [], [] ], [ [], [] ]
28  )
29 }
```


Uruchamianie i konfiguracja przepływu pracy

- Jak uruchomić:

```
1 nextflow run dna_ngs.nf \  
2   --reads 'data/*_{1,2}.fastq.gz' \  
3   --bwa_index 'ref/ref_bwa_index.{amb,ann,bwt,pac,sa}' \  
4   --outdir results \  
5   -profile docker
```

- Konfiguracja (nextflow.config):



```
1 process {  
2   cpus = 4; memory = 16.GB; time = { 1.h * task.attempt }  
3   withName: '.*:BAM_SORT_STATS_SAMTOOLS:SAMTOOLS_.*' {  
4     ext.prefix = { "${meta.id}.sorted" }  
5   }  
6   withName: '.*:BAM_SORT_STATS_SAMTOOLS:BAM_STATS_SAMTOOLS:.*' {  
7     ext.prefix = { "${meta.id}.sorted.bam" }  
8   }  
9 }  
10 params {  
11   ref_fasta = "ref/ref.fa"  
12   ref_fasta_fai = "ref/ref.fa.fai"  
13   ref_dict = "ref/ref.dict" //created by: samtools dict ref.fa -o ref.dict  
14 }  
15 docker.enabled = true  
16 docker.runOptions = '-u ${id -u}:${id -g}'
```

- Realny przykład, obrazuje praktykę pracy z Nextflow
- Kod działającego potoku przetwarzania jest dostępny pod <https://github.com/psuszyns/nf-core-dnangs>
- Jest obszerniejszy ze względu na to, że najwygodniej dodaje się moduły i subworkflowy z użyciem narzędzia `nf-core`, które dodaje dużo "boilerplate".

Podsumowanie

- Nextflow upraszcza złożone przepływy pracy w bioinformatyce dzięki DSL, kanałom i procesom.
- W porównaniu do Snakemake, Galaxy i Cromwell, Nextflow wyróżnia się skalowalnością i wsparciem społeczności.
- **nf-test** zapewnia niezawodność przepływów pracy poprzez testowanie.
- **nf-core** dostarcza gotowe, powtarzalne przepływy pracy.
- Przykład DNA NGS pokazuje łatwość praktycznego zastosowanie.

- Pytania dotyczące Nextflow, nf-test lub nf-core?
- Nextflow kontra Apache Spark
- Nextflow kontra Airflow

-  Di Tommaso, P., Chatzou, M., Floden, E. et al. *Nextflow enables reproducible computational workflows*. Nature Biotechnology 35, 316–319 (2017), DOI: 10.1038/nbt.3820.
-  Ewels, P., et al. *The nf-core framework for community-curated bioinformatics pipelines*. Nature Biotechnology, 38, 276–278 (2020), DOI: 10.1038/s41587-020-0439-x.