Introduzione a Nextflow

Principali Componenti e integrazione con k8s

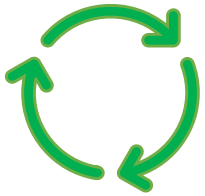
Letizia Magenta, Jacopo Gasparetto

Workshop on
management of distributed resources for genomic communities

29-30 October 2024

NEXTFLOW

Nextflow consente flussi di lavoro scientifici scalabili e riproducibili utilizzando contenitori software. Permette l'adattamento di pipeline scritte nei più comuni linguaggi di scripting. Principali caratteristiche:

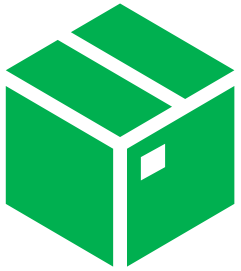


CONTROLLI CONTINUI
Tutti i risultati intermedi prodotti durante l'esecuzione vengono tracciati automaticamente.

PROTOTIPAZIONE VELOCE
Possibilità di riutilizzare script esistenti senza dover imparare un nuovo linguaggio.

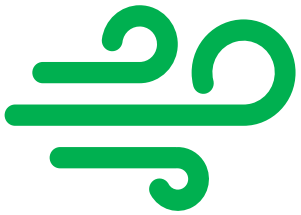
PORTABILE
Nextflow può essere eseguito su più piattaforme senza necessità di cambiare il codice (SLURM, K8S, HTC Condor, ecc.)

NEXTFLOW



RIPRODUCIBILITÀ

Nextflow supporta l'utilizzo di container (Docker o Singularity) e l'integrazione con GitHub, consentendo di scrivere pipeline CI/CD e gestire le versioni.



STREAM ORIENTED

La gestione di Nextflow si basa su flussi di dati in particolare I canali (code di elementi). I processi e I flussi di lavoro sono programmati in DSL (domain-specific language) fornito da Nextflow, basato su Apache Groovy.

PARALLELISMO

La parallelizzazione è implicitamente definita nelle dipendenze degli input e output dei vari processi.

GLOSSARIO

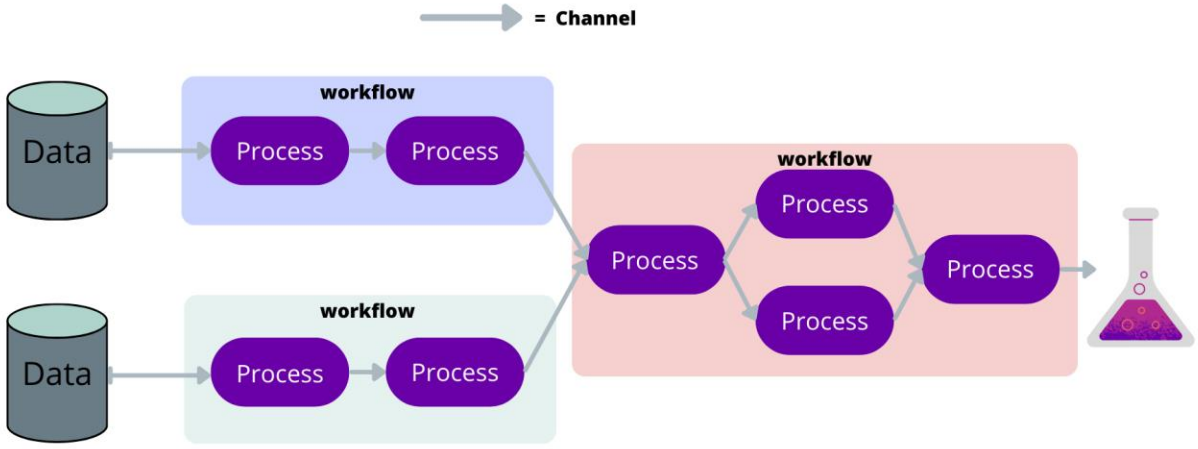
Nextflow è un software open-source progettato per semplificare l'esecuzione di flussi di lavoro bioinformatici.

WORKFLOW: definiti come script, in linguaggio Groovy, dove si specificano le diverse fasi (process) del flusso di lavoro (workflow), le dipendenze tra questi e i comandi da eseguire.

PROCESSI: unità fondamentali di Nextflow che **rappresentano le attività (TASK)** eseguibili all'interno workflow. Ogni processo ha i propri input, output e comandi associati.

CHANNELS: code utilizzate per **gestire il passaggio** di dati tra i processi. Funzionano come stream di dati, permettendo di creare pipeline in modo flessibile e di elaborare dati in parallelo.

- **Metodi**: funzioni che possono creare canali;
- **Operatori:** metodi che consumano e producono canali.



TIPI DI CANALI

QUEUE CHANNEL

Sono code **FIFO** (first-in first-out) unidirezionali che collegano un processo ***producer*** (che cioè emette un output) a un processo ***consumer*** (che consuma la coda).

Un canale può essere creato mediante:

- **Metodi** (esempio of, fromPath ecc.)

```
MyFileChannel =
channel.fromPath('/input/data/*.txt')
```

Questo file emette tanti Path quanti file con l'estensione txt sono presenti in /input/data

- **Operatori** (map, flatMap ecc.)

```
Channel.of( 1, 2, 3 )
    .flatMap { n -> [ n, n*2, n*3 ] }
    .view()
```

- **Processi** (Output)

VALUE CHANNEL

Può essere assegnato ad **uno ed un solo valore** e può essere utilizzato un numero qualsiasi di volte da un processo o da un operatore ("costante", ad es pi).

Un canale del valore può essere creato con:

- **Metodo Value**:

```
MyValue1=Channel.value( 'Hello there' )
MyValue2=Channel.value([ 1, 2, 3, 4, 5 ])
```

- **Operatori**: qualsiasi operatore che produca un singolo valore (first, collect, reduce, ecc.);

```
Channel.of( 1, 2, 3, 4 ).collect().view()
//output [ 1, 2, 3, 4 ]
```

- **Processi**: un p. emetterà value channel se viene invocato con tutti i value channel, inclusi i *parametri* che sono implicitamente racchiusi in un value channel.

PROCESSI

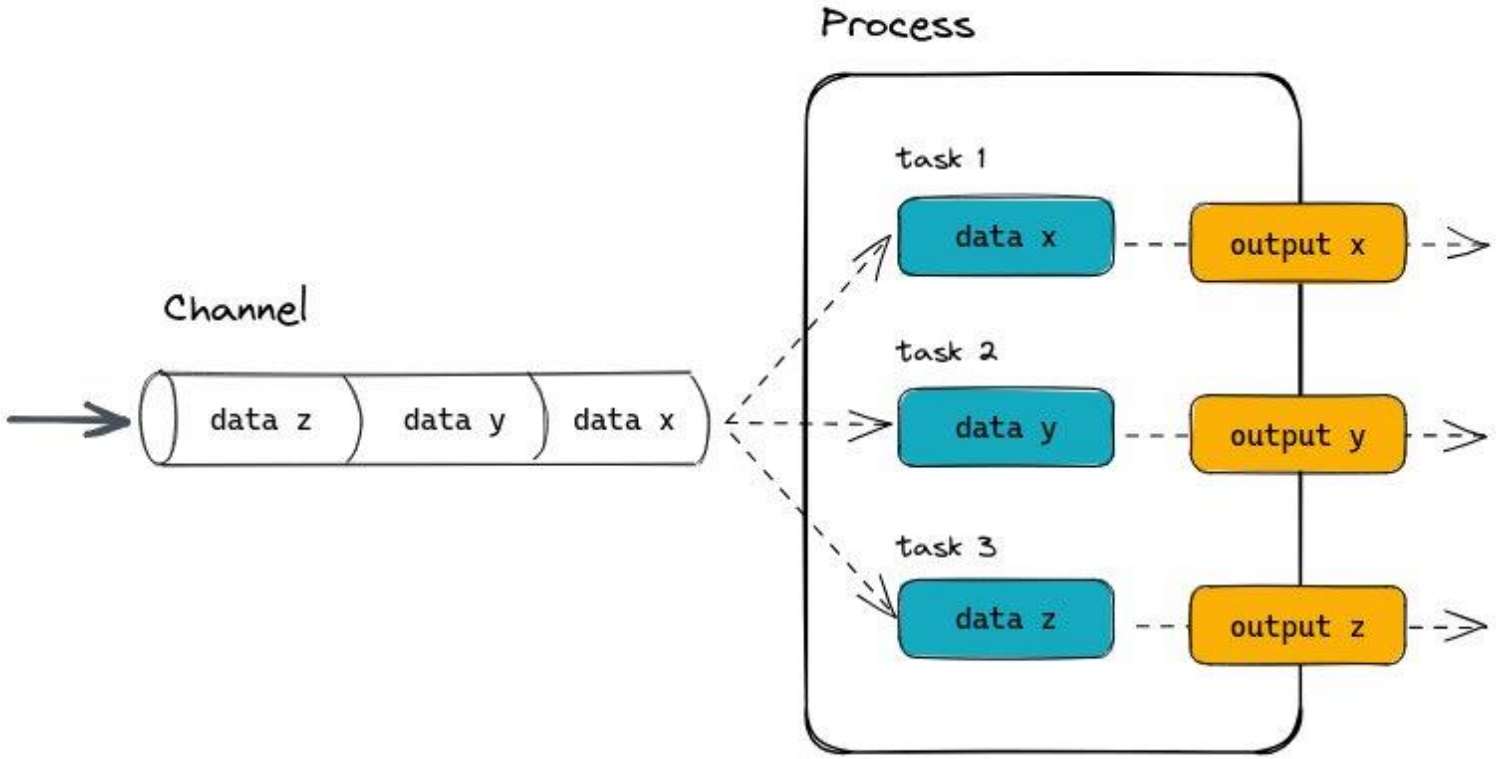
```
1 // main.nf
2
3 // data
4 // |_alice.txt
5 // |_bob.txt
6 // |_charles.txt!
7
8 process Greetings {
9   input:
10    path myFile
11  output:
12    path "greetings.txt"
13  shell:
14  '''
15  user_name=$(cat !{myFile})
16  echo "Hello, '$user_name!'" >> greetings.txt
17  '''
18  stub:
19  """
20  touch greetings.txt
21  """
22 }
23
24 workflow{
25   def inputs = Channel.fromPath("data/*.txt")
26   inputs | view              //[alice.txt, bob.txt, charles.txt]
27   Greetings(inputs) | view   //["Hello, Alice!", "Hello, Bob!", "Hello, Charles!"]
28 }
```

Il workflow viene realizzato unendo diversi processi. Quest'ultimi sono composti da diversi blocchi (statement).

- Input (del tipo: <input qualifier> <input name>)
- Output
- (Container)
- Shell
  - Ogni processo può essere scritto in **qualsiasi linguaggio di scripting** eseguibile dalla piattaforma Linux (Bash, Perl, Ruby, Python, ecc.).
  - Lo script del processo viene interpretato da Nextflow come uno **script Bash per impostazione predefinita**, ma non sei limitato a Bash.
  - Per utilizzare un linguaggio diverso da Bash, basta avviare semplicemente lo script del processo con lo **shebang** corrispondente.
  - La sezione definita come `shell` (invece di `script`) consente di definire con una **sintassi diversa** le **variabili** di Nextflow `!{varNext}` da quelle Bash `$varBash`
- (Stub (`flag -stub`))

PROCESSI

Le istanze di una processo (**TASKS**) vengono eseguite in modo i**ndipendente e isolate** le une dalle altre ma possono comunicare tra di loro tramite **code FIFO *asincrone*.**



- I **canali possono emettere dati in modo continuo**, permettendo ai processi di ricevere input non appena disponibili e avviarsi senza attese.
- Qualsiasi processo può definire uno o più canali come input e output.
- L'**interazione** tra questi processi e, in definitiva, il flusso di esecuzione (anche il ***parallelismo***) della pipeline stesso, è definita implicitamente da queste dichiarazioni di input e output.

CACHING AND RESUMING

TASK CACHE

- Nextflow memorizza nella cache le esecuzioni dei task (**`.nextflow/cache`**).
- La task cache permette di riutilizzare i risultati delle task già eseguite, evitando di rieseguire calcoli o elaborazioni già completati se i dati di input non sono cambiati.
- La ripristinabilità è utile sia per il ripristino da errori sia per lo sviluppo iterativo di una pipeline. Si può abilitare questo ripristino con la flag **`-resume`** quando si lancia la pipeline nextflow run
- Tutte le esecuzioni dei task vengono salvate automaticamente nella task cache, indipendentemente dall'opzione **`-resume`**

TASK HASH

- Nextflow calcola un hash per ogni task prima che venga eseguito.
- L'hash è un codice univoco calcolato da un seria di metadati come per esempio: il nome del task, il nome del container, gli input, gli script ecc.
- Se la ripristinabilità è abilitata (`-resume`) ed è presente una voce nella task cache *con lo stesso hash*, Nextflow tenta di ripristinare l'esecuzione dall'attività precedente.

https://www.nextflow.io/docs/latest/cache-and-resume.html#caching-and-resuming

WORKING DIRECTORY

- Vengono archiviati tutti i file prodotti durante l'esecuzione della pipeline.

1. È importante preservare sia la task cache (`.nextflow/cache`) che la cartella di lavoro (`work/<relativo-hash>`) al fine di abilitare il `resume` correttamente (controllo sui metadati, output e codice di uscita) .
2. La working directory lavora tramite softlink.
3. Serve tenere sotto controllo la working directory e pulirla quando possibile, perché tenderà ad avere grosse dimensioni. La sfruttiamo fino a quando dobbiamo fare resume della cache, o girare più volte lo stesso workflow.

```
⌥⌘2                    rocky@bastion:/work/nextflow/scratch/50/46af89ff43936dfaa77704bd2a5540
total 93G
drwxr-xr-x. 2 rocky rocky 4.0K Sep 26 23:31 .
drwxr-xr-x. 3 rocky rocky 4.0K Sep 26 22:38 ..
-rw-r--r--. 1 rocky rocky    0 Sep 26 22:38 .command.begin
-rw-r--r--. 1 rocky rocky  455 Sep 26 23:15 .command.err
-rw-r--r--. 1 rocky rocky    0 Sep 26 22:38 .command.out
-rw-r--r--. 1 rocky rocky 3.3K Sep 26 22:38 .command.run
-rw-r--r--. 1 rocky rocky  130 Sep 26 22:38 .command.sh
-rw-r--r--. 1 rocky rocky 1.1K Sep 26 22:38 .command.yaml
-rw-r--r--. 1 rocky rocky    1 Sep 26 23:31 .exitcode
lrwxrwxrwx. 1 rocky rocky   94 Sep 26 22:38 NA24385.fixmate.bam -> /work/nextflow/scratch/ac/d70e5ee182c78bbb8a0ab4f8fab8d7/NA24385_CL100076190_L2_NA.fixmate.bam
lrwxrwxrwx. 1 rocky rocky   81 Sep 26 22:38 NA24385.sort.fixmate.bam -> /work/nextflow/scratch/5a/8faa6e5c709ff93fb53e070b310bca/NA24385.sort.fixmate.bam
lrwxrwxrwx. 1 rocky rocky   85 Sep 26 22:38 NA24385.sort.fixmate.bam.bai -> /work/nextflow/scratch/5a/8faa6e5c709ff93fb53e070b310bca/NA24385.sort.fixmate.bam.bai
-rw-r--r--. 1 rocky rocky  93G Sep 26 23:14 NA24385.sort.markdup.bam
-rw-r--r--. 1 rocky rocky 9.0M Sep 26 23:31 NA24385.sort.markdup.bam.bai
[rocky@bastion 46af89ff43936dfaa77704bd2a5540]$
```

FILE DI CONFIGURAZIONE

```
// Params
params.data
params.output

// Runners
docker.enabled = true

// Profiles
profiles  {
  standard {
    process.executor = 'local'
  }

  k8s {
    process.executor = 'k8s'
  }

  my-profile {
    params.foo = "foo"
    params.baz = "baz"
    process.executor = "k8s"
  }
}

// Kuberetes
k8s.pod = [
  [volumeClaim: "pvc-input", mountPath: "/input/data"],
  [volumeClaim: "pvc-work", mountPath: "/work/nextflow"],
  [volumeClaim: "pvc-output", mountPath: "/output/results"],
]

k8s {
  namespace = "nextflow"
  context = "sorsola"
  serviceAccount = "nextflow-sa"
  debug.yaml = true
}

```

Le configurazioni del workflow sono definite in un file chiamato nextflow.config posizionato all'interno della cartella di progetto. Nextflow supporta una moltitudine di opzioni, tra cui:

- **Parametri custom** (sostituibili al momento dell'esecuzione tramite flag, ad es. **`--data foo`**);
- **L'executor da utilizzare** (ad es. k8s). L'executor è un componente che determina l'infrastruttura in cui la pipeline verrà eseguita.
- **Diversi tipi di profili** (un set di attributi che possono essere selezionati al momento dell'esecuzione della pipeline utilizzando la flag **`-profile`** (es `-profile k8s`))
- **Le personalizzazioni dei parametri dell'executor** (ad es. persistentVolumeClaim di k8s)

https://www.nextflow.io/docs/latest/reference/config.html

FILE DI CONFIGURAZIONE

Nel file di configurazione possiamo **customizzare** puntualmente **ogni singolo processo e/o pod**.

Con la keyword **`withName`** è possibile personalizzare il singolo processo.

- Ad esempio, è stato configurato il parametro "**`nodeSelector`**" (specifico di k8s) assegnando la tag `'instancetype=large'`.

```
1  // nextflow.config
2
3  // Other params and configuration
4
5  process {
6    withName: BwaMap {
7      pod { nodeSelector = 'instancetype=large' }
8    }
9
10   withName: Foo {
11     // ...
12   }
13 }
14
```

LANCIARE UN WORKFLOW

- **Utilizzo**:

```
# nextflow run <directory-workflow> -native-param-1 foo –native-param-2 baz
--custom-parm-1 bar
```

- Es:

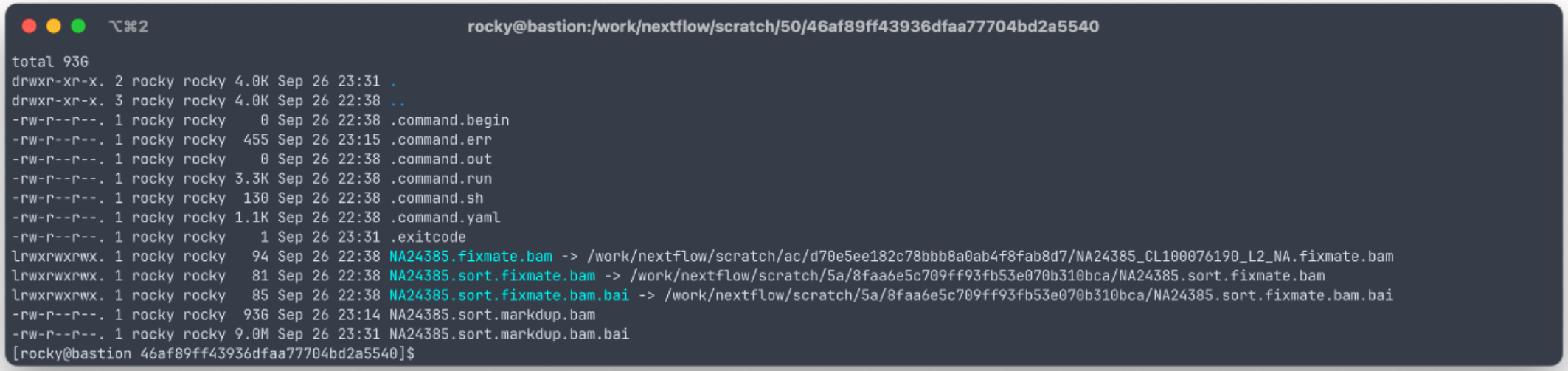
```
# nextflow run my-workflow --data ./data --output results –w work-dir
```

- Si può utilizzare al posto di `<directory-workflow>` direttamente un **repository git**, ad esempio:

```
# nextflow run sorsola/whole-genomic-sequencing --data ./data --output results
–w work-dir
```

- In questo caso Nextflow utilizza in automatico il repository "`sorsola/whole-genomic-sequencing`". La configurazione per utilizzare il gitlab privato di sorsola-epic è già disponibile su k8s.
  - Nextflow cerca di default i repository su GitHub, esempio `github.com/sorsola/whole-genomic-sequencing`.
  - È possibile istruire Nextflow per utilizzare un repository privato (`-hub`) (https://www.nextflow.io/docs/latest/git.html).

TROUBLESHOOTING



.command.sh: traduce il comando lanciato andando a sostituire con il reale contenuto le variabili di Nextflow

.command.err: con 0 o 1 segnala se il task si è concluso correttamente

.command.log: log riferiti al task

```
// process
user_name=$(cat !{myFile})
echo "Hello, '${user_name}'!" >> greetings.txt
```

```
// .command.sh
user_name=$(cat alice.txt)
echo "Hello, '${user_name}'!" >> greetings.txt
```

https://www.nextflow.io/docs/latest/cache-and-resume.html#troubleshooting

Nextflow

Integrazione con k8s

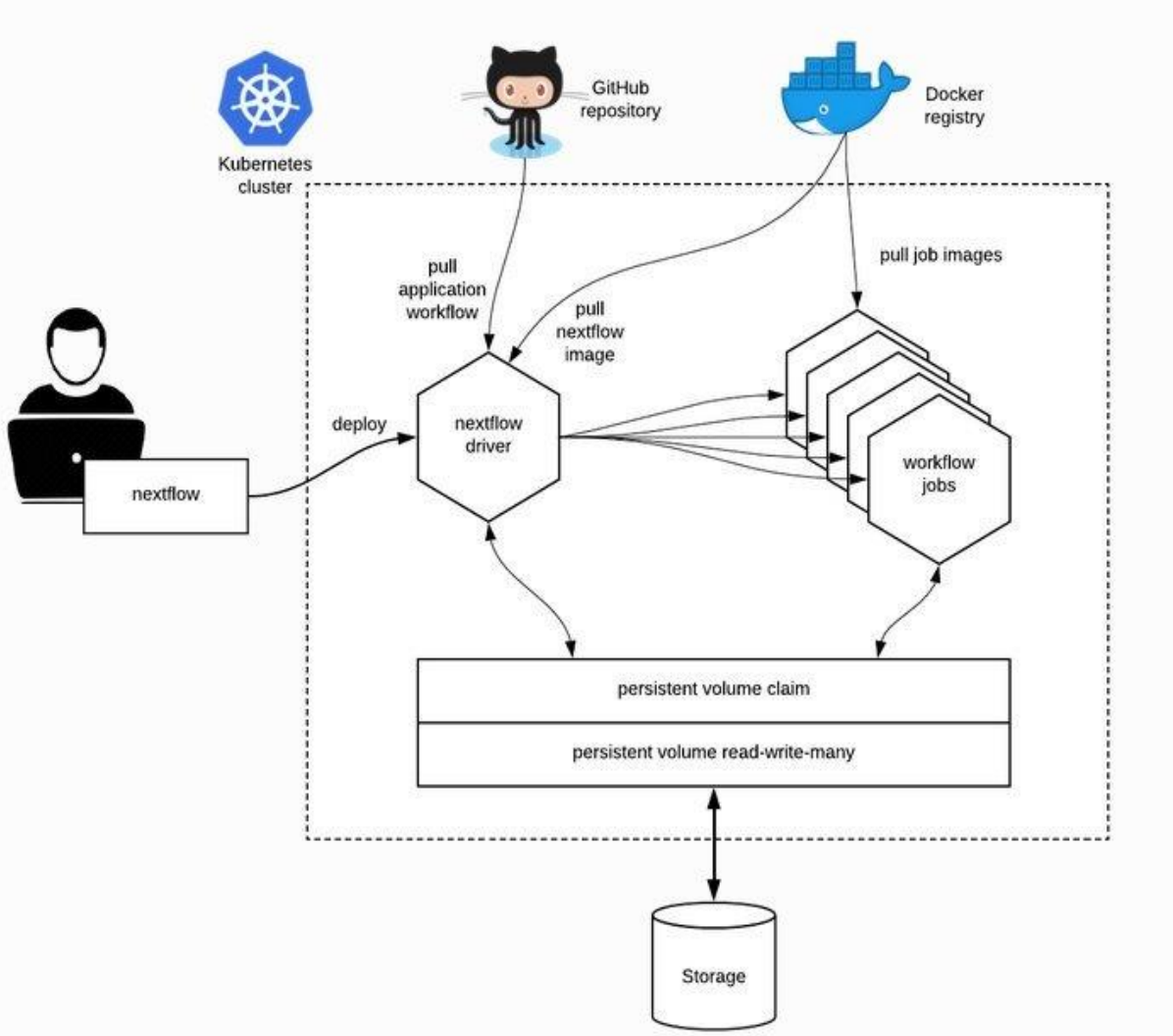
NEXTFLOW CON KUBERNETES

- Nextflow supporta nativamente Kubernetes l'esecuzione di workflow containerizzati.

- La principale astrazione di Kubernetes sono i pod. Un pod definisce lo stato desiderato di uno o più container (es. storage, network, ecc.).

- Kubernetes astrae il concetto di storage attraverso la definizione di persistent volumes che consentono ai container di accedere allo storage sottostante. Questo aspetto è **cruciale** per la condivisione dei file intermedi contenuti nella directory **`work`**.

- Un workflow può essere lanciato via command line usando il comando `nextflow`, oppure attraverso un "**pod driver**". In entrambi i casi Nexftlow creerà un pod per ogni job del workflow.



Nextflow vs Snakemake

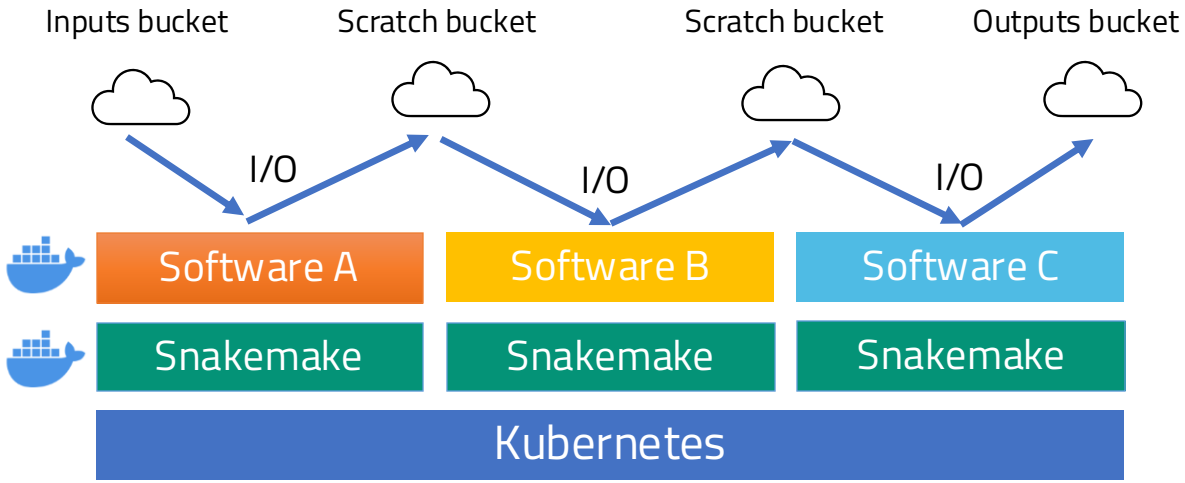
- Pipeline scritte in Python
- Incentrato su Conda Environment
- Supporto limitato ai container (container-in-container)
- Cache delle immagini per K8s non supportata
- Storage and I/O vincolati a trasferimenti S3/SFTP
- K8s PVC non supportati
- Ogni pod deve scaricare/caricare centinaia di GB di dati per ogni step del workflow

- Pipeline scritte in Groovy
- Container pienamente supportati
- Supporto nativo per i container
- Possibilità di eseguire il workflow in locale o sul cluster senza alcuna modifica
- K8s «cache friendly»
- K8s PVC supportati
- Dati di I/O immediatamente visibili ai pod/step successivi senza la necessità di trasferimenti aggiuntivi

Perché Nextflow?

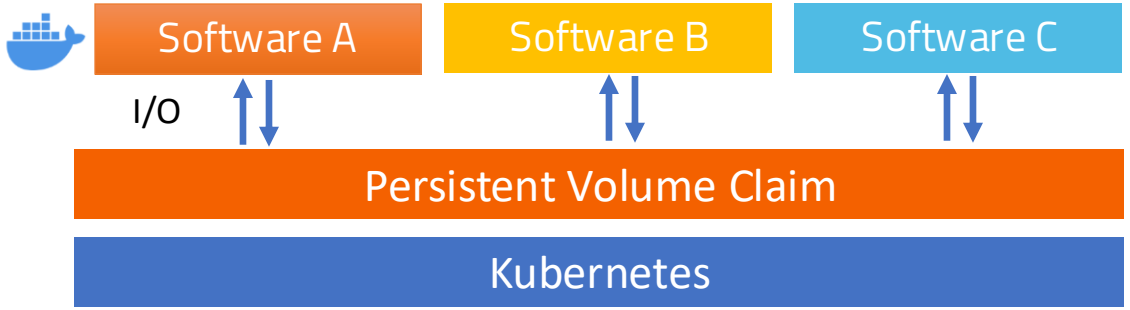
snakemake

- File scaricati/caricati verso storage S3-like
- Tutti i pod creati a partire dalla stessa immagine contenente Snakemake
- Il software finale viene eseguito come container-in-a-container (no cache da parte di K8s)



nextflow

- PVC configurati come volumi NFS montati su tutti i worker node e visibili ai pod
- File di I/O direttamente disponibili ai pod senza trasferimenti
- Pod creati a partire da immagini «native» e altamente ottimizzate (cache K8s)



Pod driver

- **`metadata.name`**: nome del pod driver
- **`metada.namespace`**: namespace da utilizzare
- **`serviceAccountName`**: account di servizio che permette al pod driver di creare e gestire i pod del workflow
- **`image`**: immagine custom di nextflow per k8s
- **`workingDir`**: destinazione di `./nextflow/cache`
- **`args`**: comando da eseguire
- **`env`**: variabili d'ambiente esposte al pod driver
- **`volume mounts`**: path dove i volumi verrano montati
- **`volumes`**: definizione dei volumi e match con i persistent volume claims e config maps

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: whole-genomic-sequencing
  namespace: nextflow
spec:
  serviceAccountName: nextflow-sa
  restartPolicy: Never
  containers:
    - name: nextflow
      image: gitlab-sorsola-integration.cnaf.infn.it:4567/sorsola/docker-images/nextflow
      imagePullPolicy: Always
      workingDir: "/work/nextflow/"
      args: [
        "nextflow", "run", "sorsola/whole-genomic-sequencing",
        "--namespace", "nextflow",
        "--serviceAccount", "nextflow-sa",
        "--context", "sorsola",
        "-profile", "k8s",
        "-w", "/work/nextflow/wgs",
        "--data", "/input/data/large",
        "--outDir", "/output/results/whole-genomic-sequencing",
        "--thBwa", "40",
        "--thSamBwa", "40",
        "--thFixmate", "40",
        "--thSamMerge", "40",
        "--thSamMark", "40",
        "--threads", "8",
        "-resume",
        "-hub", "sorsola",
        "-r", "main"
      ]
      env:
        - name: NXF_SCM_FILE
          value: "/etc/nextflow/scm"
      volumeMounts:
        - name: input-vol
          mountPath: "/input/data"
        - name: work-vol
          mountPath: "/work/nextflow"
        - name: output-vol
          mountPath: "/output/results"
        - name: gitlab-sorsola-vol
          mountPath: "/etc/nextflow"
  volumes:
    - name: input-vol
      persistentVolumeClaim:
        claimName: pvc-input
    - name: output-vol
      persistentVolumeClaim:
        claimName: pvc-output
    - name: work-vol
      persistentVolumeClaim:
        claimName: pvc-work
    - name: gitlab-sorsola-vol
      configMap:
        name: gitlab-sorsola
        items:
          - key: providers
            path: scm
```

Pod driver

- **metadata**:
  - **name**: friendly name del pod
  - **namespace**: namespace da utilizzare

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nextflow-driver
  namespace: nextflow
spec:
  # ...

```

Pod driver

- **`serviceAccountName`**: account di servizio per dare al driver i permessi di creare e gestire gli ulteriori pod e pullare le immagini dal registry privato di gitlab-sorsola-integration
- **`containers`**: andiamo a definire un solo container basato sull'immagine
- **`args`**: commando nextflow da eseguire
  - **`-hub`** sorsola indica il provider git custom (gitlab, vedi in seguito)
  - **`-r main`** indica la branch (revision) git da utilizzare

```yaml
# ...
spec:
  serviceAccountName: nextflow-sa
  restartPolicy: Never
  containers:
    - name: nextflow
      image: gitlab-sorsola-integration.cnaf.infn.it:4567/sorsola/docker-images/nextflow
      imagePullPolicy: Always
      args: [
        "nextflow", "run", "sorsola/whole-genomic-sequencing",
        "--namespace", "nextflow",
        "--serviceAccount", "nextflow-sa",
        "--context", "sorsola",
        "-profile", "k8s",
        "-w", "/work/nextflow/scratch",
        "--data", "/input/data/my-experiment",
        "--outDir", "/output/results/my-experiment",
        "--thBwa", "40",
        "--thSamBwa", "40",
        "--thFixmate", "40",
        "--thSamMerge", "40",
        "--thSamMark", "40",
        "--threads", "8",
        "-resume",
        "-hub", "sorsola",
        "-r", "main"
      ]
#...
```

Pod driver

- **env**: variabili d'ambiente da esporre al pod driver
  - **`NXF_SCM_FILE`** indica dov'è locato il file contentente i provider custom git da dove pullare il codice del workflow
- **`volumeMounts`**: path dove vengono montati i volumi relativi a
  - input
  - work directory
  - output
  - scm file contente la config per gitlab. In questo caso verrà montanto in `/etc/nextflow/scm`

```yaml
# ...
spec:
  # ...
  containers:
    - name: nextflow
      # ...
      env:
        - name: NXF_SCM_FILE
          value: "/etc/nextflow/scm"
      volumeMounts:
        - name: input-vol
          mountPath: "/input/data"
        - name: work-vol
          mountPath: "/work/nextflow"
        - name: output-vol
          mountPath: "/output/results"
        - name: gitlab-sorsola-vol
          mountPath: "/etc/nextflow"
       # ...

```

Pod driver

- **`volumes:`** definizione della tipologia di volume, ad esempio *persistantVolumeClaim* o *configMap*

In questo caso, la **`configMap`** `gitlab-sorsola` rappresenta un file salvato all'interno di k8s, contente la configurazione accedere a https://gitlab-sorsola-integration.cnaf.infn.it

```yaml
# ...
spec:
  # ...
  containers:
    # ...
  volumes:
    - name: input-vol
      persistentVolumeClaim:
        claimName: pvc-input
    - name: output-vol
      persistentVolumeClaim:
        claimName: pvc-output
    - name: work-vol
      persistentVolumeClaim:
        claimName: pvc-work
    - name: gitlab-sorsola-vol
      configMap:
        name: gitlab-sorsola
        items:
          - key: providers
            path: scm

```

Pod driver

La **configMap** `gitlab-sorsola` rappresenta un file salvato all'interno di k8s, contente la configurazione accedere a GitLab. È a tutti gli effetti un file di resto il cui contenuto è raffigurato in verde. La chiave `providers` a riga n.7 viene utilizzata alla riga n.20 della slide precedente.

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: gitlab-sorsola
5    namespace: nextflow
6  data:
7    providers: |
8      providers {
9        sorsola {
10         server = 'https://gitlab-sorsola-integration.cnaf.infn.it'
11         platform = 'gitlab'
12         user = 'read-repository'
13         password = 'xxxxxxxxxxxxxxx'
14       }
15     }
16
```