The Nextflow Automation Blueprint

Before writing code, you need the engine and the garage.

1. **Install Nextflow:** The "engine" that runs the script.
2. **Install Docker:** The "garage" that holds all the tools (FastQC, BWA, etc.) so you don't have to install them on your actual computer.
3. **Organize Folders:** Create a project root with a modules/ folder and a data/ folder.

The Role of Containerization in Bioinformatics
Abstract Analogy: The "Shipping Container" Model
Docker utilizes a **containerization** strategy analogous to global intermodal shipping. Just as a standard shipping container encapsulates diverse goods—ensuring they can be transported via ship, rail, or truck without altering the internal contents—a Docker container packages bioinformatics software with its entire runtime environment. This ensures that the application remains functional and consistent regardless of the host operating system.

The Problem: Dependency Hell and Software Conflicts
In complex genomic pipelines, different tools often require mutually exclusive environments. For example:

- **Tool A (e.g., BWA):** May rely on legacy libraries or specific versions of Python (e.g., 2.7).
- **Tool B (e.g., MultiQC):** May require modern dependencies (e.g., Python 3.10+).

Installing these directly on a single host machine creates **dependency conflicts**, where updating a library for one tool inadvertently breaks another. Docker resolves this by providing **process isolation**; each tool resides in its own discrete environment, preventing cross-tool interference.

Core Definitions: Images vs. Containers

Term	Technical Definition	Practical Analogy
Docker Image	A read-only, executable package that includes everything needed to run an application—code, runtime, libraries, and environment variables.	The **Blueprint**: A static template used to build the environment.
Docker Container	A runtime instance of an image. It is a lightweight, standalone, and executable software package that runs in isolation on the host OS.	The **Live Instance**: The active "worker" that executes the task and is decommissioned upon completion.

Conclusion: Why Containerization is Mandatory

For high-throughput sequencing pipelines, Docker is utilized to achieve:

1. **Reproducibility:** Ensuring that the same code produces identical results across different computational infrastructures.
2. **Portability:** Enabling the seamless transition of workflows from local workstations to high-performance computing (HPC) clusters or cloud environments.
3. **Efficiency:** Eliminating the time-intensive process of manual tool installation and configuration.

Phase 2: Component Engineering (The Modules)

For every tool in your pipeline, you create a dedicated .nf file in the modules/ directory.

- **Identify the Tool:** (e.g., Samtools).
- **Find the Container:** Search Biocontainers for the Docker URL.
- **Define Inputs/Outputs:** Decide what file comes in (FASTQ) and what file goes out (BAM).
- **Write the Script:** The exact command you would type in a terminal.

Phase 3: The Brain (The Main Workflow)

Create the main.nf file to act as the conductor.

1. **Define Parameters:** Set default paths for inputs (params.fastq) and outputs (params.outdir).
2. **Import Modules:** Use the include statement to bring in your module files.
3. **Construct Channels:** Use Channel.fromPath to feed your files into the pipeline.
4. **Connect the Dots:** Pass the output of one process as the input to the next (e.g., TRIM(FASTQC.out)).

Phase 4: Configuration & Safety

1. **nextflow.config:** Create this file to tell Nextflow to "Always use Docker."
   docker.enabled = TRUE

2. **.gitignore:** The purpose of a .gitignore file is to act as a **filter**. It tells Git exactly which files or folders in your project should be **ignored** (not tracked or uploaded to GitHub).

   ```
   .nextflow/        //  --internal cache files that are only useful on local machine
   .nextflow.log*    //  --unnecessary log files
   work/             //  --contain 100s of temporary folders not needed in GitHub
   results/          //  --contains outputs
   data/             //  --contains raw FASTQ files
   reference/        //  --contains raw FASTQ files
   ```

1. **Git Init:** Initialize your repository.
2. **Commit:** Save a "snapshot" of your code.
3. **GitHub Push:** Upload your code (using a Token) so it's safe in the cloud.

Pipeline Execution Flowchart

Step	Action	Input	Output
1	**Quality Control**	Raw FASTQ	HTML Report
2	**Trimming**	Raw FASTQ	Cleaned FASTQ
3	**Alignment**	Cleaned FASTQ + Ref	SAM File
4	**Processing**	SAM File	Sorted BAM + Index
5	**Variant Calling**	Sorted BAM + Ref	**VCF File**
6	**Reporting**	All QC Logs	**MultiQC Report**

The "From Scratch" Checklist

If you were to start a brand new project tomorrow, follow this order:

1. Initialize Git (git init).
2. Create .gitignore (Block work/ and data/).
3. Write a single module (e.g., fastqc.nf).
4. Write a basic main.nf to test that one module.
5. Run with nextflow run main.nf -with-docker.
6. Add more modules one by one until the pipeline is complete.
7. Update the README.md.
8. Push to GitHub.

Nextflow DSL2 Core Cheat Sheet

Every step (module) follows this 5-part structure:

```
Code snippet

process NAME {
    container '...'         // Logic: Where is the software?
    publishDir '...', ...   // Logic: Where do results go?

    input:
    path x                  // Logic: What do I need?

    output:
    path '...'              // Logic: What am I making?

    script:
    """
    command $x              // Logic: How do I run the tool?
    """
}
```

- **The container line (The Registry)**

Bioinformatics has a massive "library" of pre-made containers called **Biocontainers**, the container directive is used to link a process to a Docker image.

> **Where to find them:** You go to biocontainers.pro and search for the tool (like fastqc or samtools).
> **The Pattern:** Almost all of them follow this format: <mark>quay.io/biocontainers/TOOLNAME:VERSION--BUILD.</mark>

- **The publishDir line (Your Logic)**

This doesn't come from a website; **you invent it** based on how you want to organize your results.

> **The Logic:** You ask yourself: "Where do I want the output of this specific step to go?"
> **The Pattern:** It almost always looks like this: publishDir <mark>"${params.outdir}/FOLDER_NAME", mode: 'copy'</mark>

Section	Purpose	Example Syntax
Directive	Configuration	container 'URL', publishDir 'path', cpus 2
Input	What it needs	path fastq, val sample_id
Output	What it saves	path "*.bam", path "report.html"
When	Condition	when: params.run_this_step
Script	The command	""" fastqc $fastq """

- **container**: Links to a Docker/Singularity image (look up on Biocontainers).
- **publishDir**: Moves files from the hidden work/ folder to a visible folder.
  - *Tip:* Always use mode: 'copy' so files don't disappear if you delete the work/ directory.
- **tag**: Adds a label to the logs (e.g., tag "$fastq").

3. Essential Channel Operators

Channels are the "pipes" that move data between processes.

- **Channel.fromPath('/path/*.fastq')**: How you get files into the pipeline.
- **.view()**: Prints the contents of a channel (great for debugging).
- **.mix()**: Combines two or more channels into one.
- **.collect()**: Waits for **all** items in a channel to finish before passing them as a single list (essential for **MultiQC**).

4. The Workflow Block (main.nf)

This is where you "glue" the modules together.

Code snippet

```
include { FASTQC } from './modules/qc.nf'

workflow {
    // 1. Create Input Channel
    input_ch = Channel.fromPath(params.input)

    // 2. Execute Process
    qc_results = FASTQC(input_ch)

    // 3. Use Output in another process
    // next_process(qc_results)
}
```

5. Exam Troubleshooting Logic

If your pipeline fails, check these three things:

1. **Paths:** Did you provide the right path to the file?
2. **Wildcards:** Did you use * in the output: block to catch the files?
3. **Commas:** Did you forget a comma in the publishDir or input section?

6. Helpful Linux Commands for Nextflow

- nextflow run main.nf -resume: Runs the pipeline skipping finished steps.
- nextflow log: Shows a list of your previous runs.
- nextflow clean -f -q: Cleans up the heavy work/ directory to save space.