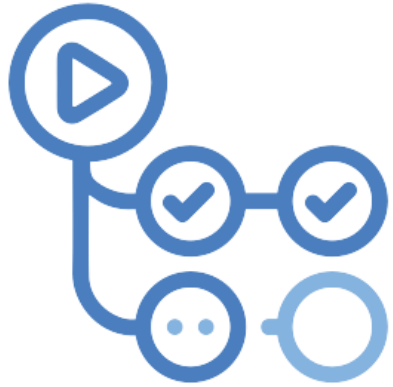
Bioinformatics Data Skills

Workflows and Docker containers for Reproducible Research



Tamposis Ioannis

Researcher & Software Development Engineer

B.Sc. in Informatics, M.Sc. in Biomedical, Ph.D. in Bioinformatics

Department of Computer Science and Biomedical Informatics, University of Thessaly

Outline

1. Why need automated workflows and Docker?

2. Go through some code

3. Hands-on project

Comparability and **Reproducibility** are nowhere to be found

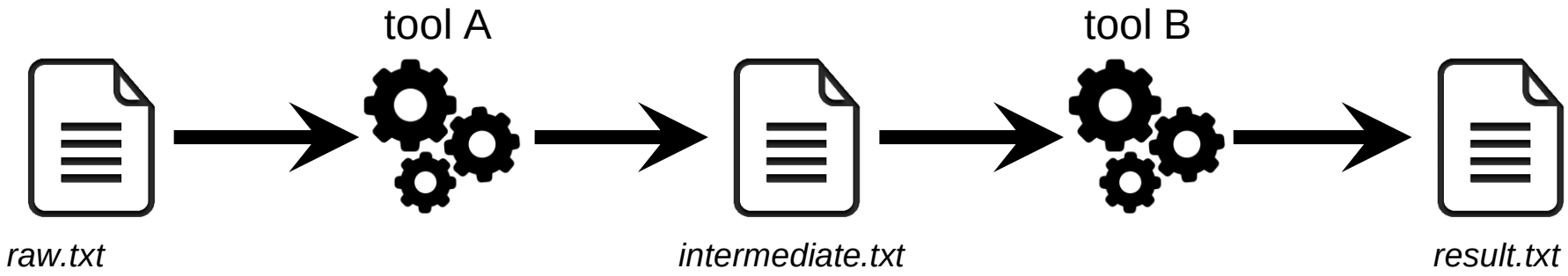
NEWS | 12 October 2023

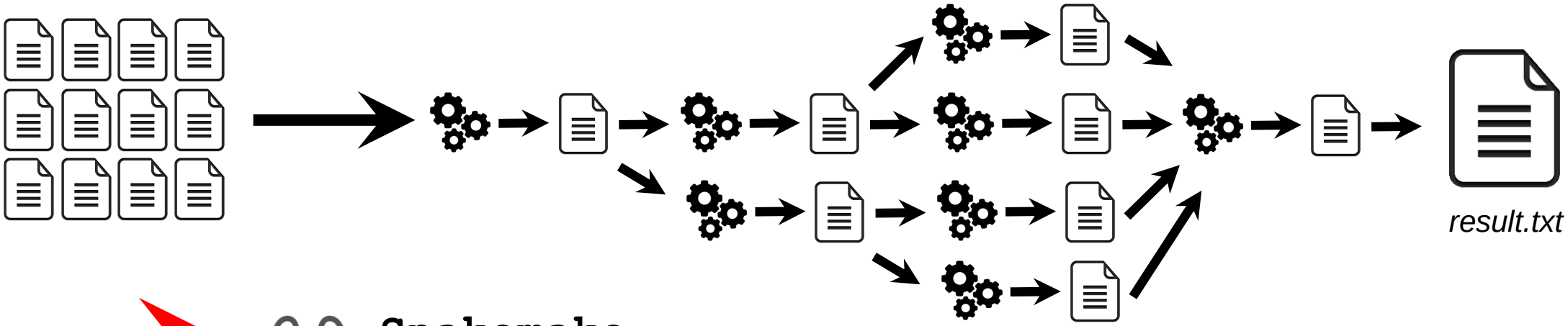
Reproducibility trial: 246 biologists get different results from same data sets

Ροές εργασιών (Workflows)

- Μια ροή εργασίας (ή αγωγός) είναι μια ακολουθία λειτουργιών που χρησιμοποιούνται για την ολοκλήρωση μιας διαδικασίας.
- Στη βιολογία, χρησιμοποιούμε συχνά τεχνολογία υψηλής απόδοσης, δημιουργώντας μεγάλο όγκο δεδομένων και χρησιμοποιούμε έναν αρκετά μεγάλο αριθμό διαφορετικών εργαλείων βιοπληροφορικής, συχνά εξειδικευμένα σε μία εργασία, για την ανάλυση των δεδομένων που προκύπτουν.
- Αυτό μπορεί να οδηγήσει στη δημιουργία πολύπλοκων ροών εργασιών ανάλυσης που είναι δύσκολο να διαχειριστούν με μη αυτόματο τρόπο.

Orchestration strategies: workflow managers





Snakemake
a Python workflow manager

Why Workflow managers?

- Τα εργαλεία διαχείρισης ροής εργασιών, όπως το Snakemake, υπάρχουν για να **ελαχιστοποιήσουν τον αριθμό των μη αυτόματων βημάτων** που απαιτούνται για την εκτέλεση μιας ροής εργασιών ανάλυσης.
- Απλοποιούν την ανάπτυξη, τη συντήρηση και τη χρήση του αγωγού, αντιμετωπίζοντας την **παραλληλοποίηση εργασιών**, την **επανάληψη αποτυχημένων εκτελέσεων ή βημάτων** και την **παρακολούθηση παραμέτρων**, για παράδειγμα, και κάνοντας απλώς τον κώδικά σας γενικά λιγότερο περίπλοκο στην ανάγνωση και αρθρωτή.
- Κάνουν επίσης τη διοχέτευσή σας πιο εύκολα επεκτάσιμη σε μεγάλα σύνολα δεδομένων.

Τι να χρησιμοποιήσω;

- Υπάρχουν πολλά εργαλεία διαχείρισης ροής εργασιών (https://github.com/pditommaso/awesome-pipeline),
- μερικά είναι πιο συγκεκριμένα για την κοινότητα της βιοπληροφορικής και συχνά συμβατά με συνήθειες εφαρμογής FAIR για να κάνουν τη ροή εργασίας πιο φορητή και αναπαραγώγιμη, όπως η χρήση containers (π.χ. apptainer/singularity, docker) ή διαχειριστές πακέτων (π.χ. conda).
- Ποιο εργαλείο διαχείρισης ροής εργασίας θα χρησιμοποιηθεί θα εξαρτηθεί από την εφαρμογή σας και τις ιδιότητες που σας ενδιαφέρουν (π.χ. ευκολία χρήσης, φορητότητα, επεκτασιμότητα, διαθεσιμότητα κ.λπ.)
- Δείτε τον Πίνακα 1 στο Wratten et al , https://www.nature.com/articles/s41592-021-01254-9/tables/1

Tool	Class	Ease of use[a]	Expressiveness[b]	Portability[c]	Scalability[d]	Learning resources[e]	Pipeline initiatives[f]
Galaxy	Graphical	●●●	●○○	●●●	●●●	●●●	●●○
KNIME	Graphical	●●●	●○○	○○○	●●◐	●●●	●●○
Nextflow	DSL	●●○	●●●	●●●	●●●	●●●	●●●
Snakemake	DSL	●●○	●●●	●●◐	●●●	●●○	●●●
GenPipes	DSL	●●○	●●●	●●○	●●○	●●○	●●○
bPipe	DSL	●●○	●●●	●●○	●●◐	●●○	●○○
Pachyderm	DSL	●●○	●●●	●○○	●●○	●●●	○○○
SciPipe	Library	●●○	●●●	○○○	○○○	●●○	○○○
Luigi	Library	●●○	●●●	●○○	●●◐	●●○	○○○
Cromwell + WDL	Execution + workflow specification	●○○	●●○	●●●	●●◐	●●○	●●○
cwltool + CWL	Execution + workflow specification	●○○	●●○	●●◐	○○○	●●●	●●○
Toil + CWL/WDL/Python	Execution + workflow specification	●○○	●●●	●◐○	●●●	●●○	●●○

Δημοφιλή παραδείγματα:

- Οι πιο συχνά χρησιμοποιούμενοι διαχειριστές ροής εργασίας στον τομέα της βιοπληροφορικής αυτή τη στιγμή είναι οι διαχειριστές ροής εργασιών με γραφικά και DSL (Γλώσσα συγκεκριμένης περιοχής).
- Οι διαχειριστές γραφικών, των οποίων ένα δημοφιλές παράδειγμα είναι το Galaxy, δεν απαιτούν εμπειρία προγραμματισμού (δηλαδή "σημείο και κλικ"). Από την άλλη πλευρά, οι διαχειριστές ροής εργασιών που βασίζονται σε DSL έχουν σχεδιαστεί για βιοπληροφορικούς επειδή απαιτούν ελάχιστη εμπειρία προγραμματισμού.
- Το Nextflow και το Snakemake είναι πολύ δημοφιλή παραδείγματα τέτοιων διαχειριστών, με τη μεγαλύτερη κοινότητα και πολλούς διαθέσιμους αγωγούς έτοιμους προς χρήση.

Workflow management systems

COMMON
WORKFLOW
LANGUAGE

nextflow

snakemake

Benefits of automated workflows

1. Increased reproducibility of analyses

2. Increased **data consistency and comparability**

3. More efficient error tracking (log files) and solving

4. Promotes best-practice bioinformatic strategies

But what is reproducibility?

No one agrees: That's actually replication / repetition / ...

Reproduce: can someone else take the data / software and get the same answer?

Replicate: can anyone get the same answer from a different study

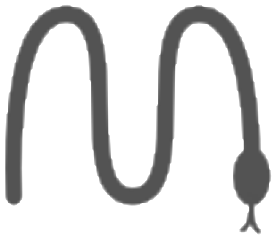
But what is reproducibility, really?

The Rs of "small r" reproducibility:

- Reproduce
- Replicate
- Reliable & robust
- Repeat & rerun
- Report
- Reuse
- Resource efficient ...

Snakemake

- Workflow management system
- Designed by Johannes Köster
- Now PI at UniEssen
- Python3 – based
- cmake philosophy
- conda installation
- conda support
- http://snakemake.readthedocs.io/



Why Snakemake?

- Συντηρείται από ακαδημαϊκό εργαστήριο και είναι **ανοιχτού κώδικα**.
- Είναι επίσης ένα από τα πιο δημοφιλή εργαλεία διαχείρισης ροής εργασίας που βασίζονται σε DSL και διαθέτει μια μεγάλη κοινότητα που παρέχει πληθώρα πόρων εκμάθησης και άμεση πρόσβαση στην υποστήριξη της κοινότητας.

Why Snakemake?

- Η γλώσσα του Snakemake είναι παρόμοια με αυτή της τυπικής σύνταξης Python
- λειτουργεί αντίστροφα ζητώντας αρχεία εξόδου και ορίζοντας κάθε βήμα που απαιτείται για την παραγωγή τους (παρόμοιο με το Make).

Why Snakemake?

- Τα υπάρχοντα εργαλεία και τα pipelines που είναι γραμμένα σε άλλες γλώσσες δέσμης ενεργειών μπορούν εύκολα να ενσωματωθούν στα Snakemake pipelines.
- Το Snakemake επιτρέπει την προσωρινή αποθήκευση μεταξύ ροής εργασιών για να αποφευχθεί ο επανυπολογισμός των κοινών βημάτων μεταξύ των pipelines.

Installation

Install miniconda
Download and run the installer (eg. `Miniconda3-latest-Linux-x86_64.sh`)

Install snakemake with conda

```
conda install -c bioconda -c conda-forge snakemake
conda install -c bioconda -c conda-forge snakemake-minimal
```

Test

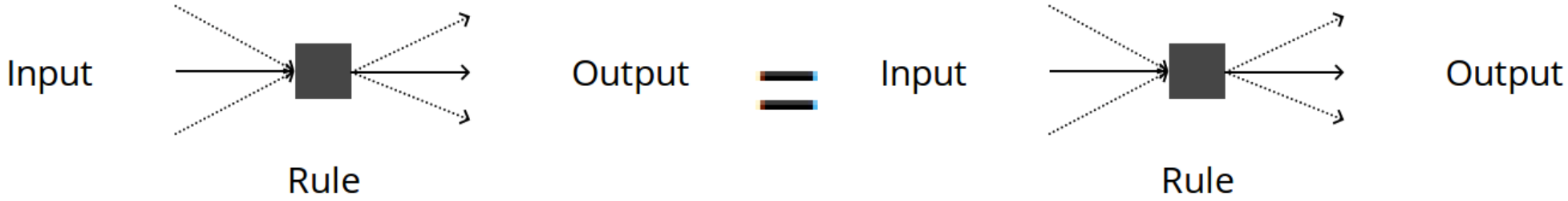
```
snakemake --version
```

How  snakemake works

- Workflows organized in sets of inter-depended rules
- Runs Python code (can work like a Python script)
- User Friendly / Extends Python functionality

How snakemake works

- Μια ροή εργασιών snakemake συνδέει κανόνες χάρη στα ονόματα αρχείων των οδηγιών εισαγωγής και εξόδου κανόνων:



Snakemake rules order:

ο πρώτος κανόνας καθορίζει τα αρχεία αποτελεσμάτων, οι επόμενοι κανόνες περιγράφουν τον τρόπο επίτευξής τους.

How snakemake works

- στο παρακάτω παράδειγμα διάγραμμα, θα ξεκινήσει ελέγχοντας εάν υπάρχουν "δεδομένα εξόδου" και εάν όχι, θα ελέγξει εάν υπάρχει "tmp data n-1" για τη δημιουργία "δεδομένων εξόδου" και ούτω καθεξής.

What’s it look like?

- It's just a Python (3) script
- A set of rules (input / output / shell or run)
- Snakemake computes dependencies
- Built-in text substitution!

```
rule wordcount_isles:
  input: "moby-dick.txt"
  output: "moby-wordcount.txt"
  shell: "wc -w {input} > {output}"
```

And how do you use it?

- Look for a file called Snakemake
- Runs the first rule in it
- Both these things can be changed

```
% snakemake
```

What does the workflow look like?

```
rule print_results:
  input: "moby-wordcount.txt"
  run:
    with open (input, 'w') as in_hndl:
      for line in in_hndl:
        print (line)

rule wordcount_isles:
  input: "moby-dick.txt"
  output: "moby-wordcount.txt"
  shell: "wc -w {input} > {output}"
```

With and without wilcards examples

```
rule all :
        input : "P10415.fasta", "P01308.fasta"
rule get_prot :
        output : "P10415.fasta", "P01308.fasta "
        run :
        shell (" wget https://www.uniprot.org/uniprot/P10415.fasta")
        shell (" wget https://www.uniprot.org/uniprot/P01308.fasta")


rule get_prot :
  output : "{prot}.fasta "
  run :
     shell(" wgethttps://www.uniprot.org/uniprot/{wildcards.prot}.fasta ")
```

snakemake

Tell Snakemake what files you want to be created

```
rule:
  input: "A.txt", "B.txt", "C.txt"
```

Produce the files you want to have from some intermediate result

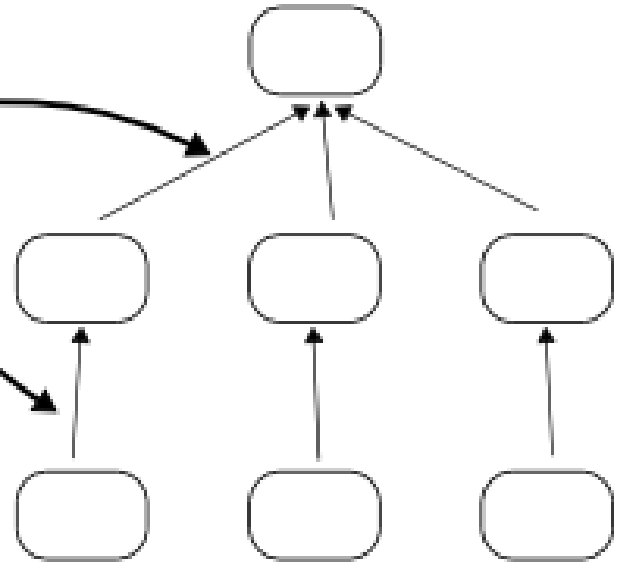
```
rule:
  input: "{sample}.inter"
  output: "{sample}.txt"
  shell: "somecommand {input} {output}"
```

Create a needed intermediate result

```
rule:
  input:  "{sample}.in"
  output: "{sample}.inter"
  run:
    somepythoncode()
```

Use wildcards to write general rules for all samples

Snakemake determines the dependencies for you



snakemake

GNU Make

Tell Snakemake what files you want to be created

```
rule:
  input: "A.txt", "B.txt", "C.txt"
```

Produce the files you want to have from some intermediate result

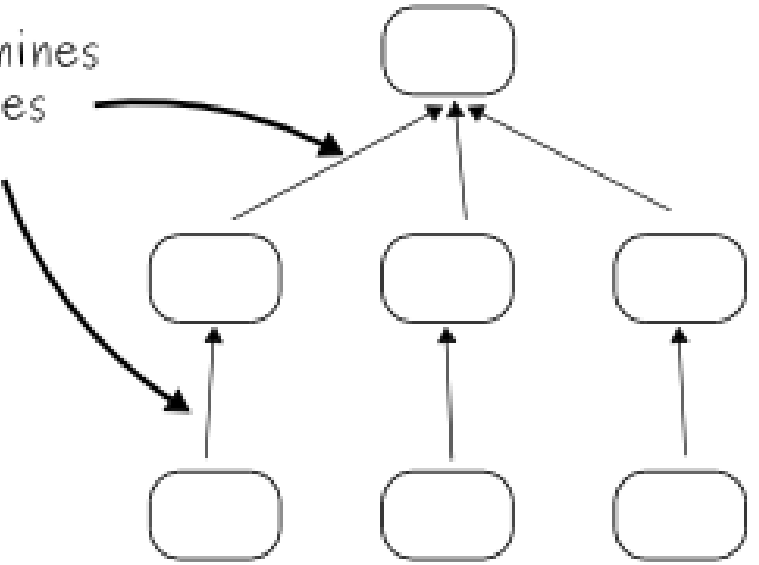
```
rule:
  input: "{sample}.inter"
  output: "{sample}.txt"
  shell: "somecommand {input} {output}"
```

Create a needed intermediate result

```
rule:
  input:  "{sample}.in"
  output: "{sample}.inter"
  run:
    somepythoncode()
```

Use wildcards to write general rules for all samples

Snakemake determines the dependencies for you



Example: mapping reads

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"
```

Example: sorting & indexing reads

```
rule samtools_sort:
  input: "mapped_reads/{sample}.bam"
  output: "sorted_reads/{sample}.bam"
  shell:
    "samtools sort -T sorted_reads/{wildcards.sample} "
    "-O bam {input} > {output}"

rule samtools_index:
  input: "sorted_reads/{sample}.bam"
  output: "sorted_reads/{sample}.bam.bai"
  shell: "samtools index {input}"
```

Example: script & keyword arguments

```
rule rewrite_files:
  input: "path/to/infile", "path/to/other/infile"
  output: first="path/to/outfile",second="path/to/other/outfil
  run:
    # write both infiles to both outfiles
    for f in input:
      ...
      with open (output.first, "w") as out:
        out.write (...)
      with open (output.second, "w") as out:
        out.write (...)
```

Reports

```
rule report:
  input: "calls/all.vcf"
  output: "report.html"
  run:
    with open (input[0]) as vcf:
      n_calls = sum (1 for l in vcf if not l.startswith("#

    report("""
    An example variant calling workflow
    ===================================

    Reads mapped to Yeast ref genome, giving {n_calls} varia
    """, output[0], T1=input[0])
```

snakemake

Workflow flow is implicitly declared through these input-output relationships

```
rule count:
    input: bams = expand('{sample}.Aligned.sortedByCoord.out.bam', sample=samples)
    threads: config['featureCounts_threads']
    output: counts='counts.txt'
    conda: "envs/rnaseq.yaml"
    params: gtf=config['gtf']
    shell:
        """ featurecounts -M
                          -s 0
                          -T {threads}
                          -p
                          -t exon
                          -g gene_id
                          -a {params.gtf}
                          -o {output.counts} {input.bams} """

rule modify:
    input: counts="counts.txt"
    output: counts_mod="counts.mod.txt"
    shell:
        """ perl counts_mod.pl {input.counts} > {output.counts_mod} """
```

snakemake

If no input is given for a rule, it runs automatically

Install tool through conda (needs *--use-conda* when run)

Threads, input fasta and gtf files can be defined in *config.yaml* file

```
rule build_genome_index:
    output: 'index_chkp'
    conda: "envs/rnaseq.yaml"
    threads: config['star_build_threads']
    message: "Building genome index"
    params:
        genome_fasta=config['genome_fasta'],
        gtf=config['gtf']
    shell:
        """ star --runThreadN {threads} \
                 --runMode genomeGenerate \
                 --genomeDir genome/  \
                 --genomeFastaFiles {params.genome_fasta} \
                 --sjdbGTFfile {params.gtf} \
                 --sjdbOverhang 149 && \

            mkdir genome/  && \
            touch index_chkp """
```

snakemake

rule all asks final output as *input*

```
rule all:
    input:
        expand("edgeR/02_analyze_DE/{de_subset}.P1e-3_C{log2FC_cutoff}.DE.annotated.sorted.xlsx",
               de_subset=de_subset,
               log2FC_cutoff=config['log2FC_cutoff'])
```

Wildcards to get names from fastq filenames

```
## Wildcards
samples = {f[:-11] for f in os.listdir(config['read_dir'] if f.endswith(".fastq.gz"))}
groups = [re.sub("\d+$|_\d+$", "", i) for i in samples]
groups = np.unique(groups)

def all_pairs(x):
    samp = (s for s in x)
    comparisons = {}

    for sample1, sample2 in itertools.combinations(samp, 2):
        comparisons[str(sample1 + "_" + sample2)] = str(sample1 + "_vs_" + sample2)

    return list(comparisons.values())

de_subset = all_pairs(groups)
wildcard_constraints: samples = "trim"
```

Python function to create the *de_subset* wildcard

rule all asks final output as *input*

```
rule all:
    input:
        expand("edgeR/02_analyze_DE/{de_subset}.P1e-3_C{log2FC_cutoff}.DE.annotated.sorted.xlsx",
               de_subset=de_subset,
               log2FC_cutoff=config['log2FC_cutoff'])
```

Wildcards to get names from fastq filenames

```
## Wildcards
samples = {f[:-11] for f in os.listdir(config['read_dir'] if f.endswith(".fastq.gz"))}
groups = [re.sub("\d+$|_\d+$", "", i) for i in samples]
groups = np.unique(groups)

def all_pairs(x):
    samp = (s for s in x)
    comparisons = {}

    for sample1, sample2 in itertools.combinations(samp, 2):
        comparisons[str(sample1 + "_" + sample2)] = str(sample1 + "_vs_" + sample2)

    return list(comparisons.values())

de_subset = all_pairs(groups)
wildcard_constraints: samples = "trim"
```

Python function to create the *de_subset* wildcard

Conda environments, defined in a *yaml* file

```
name: rnaseq
channels:
    - conda-forge
    - bioconda
dependencies:
    - fastqc
    - trimmomatic
    - star
    - subread
    - edgeR
```

```
read_dir: 'name of directory w
trimmomatic_threads: 12
genome_fasta: "genome.fasta"
gtf: "genome.gtf"
star_build_threads: 12
star_map_threads: 20
featureCounts_threads: 12
log2FC_cutoff: 2
```

Parameters defined in a *config.yaml* file

snakemake

rule all asks final output as *input*

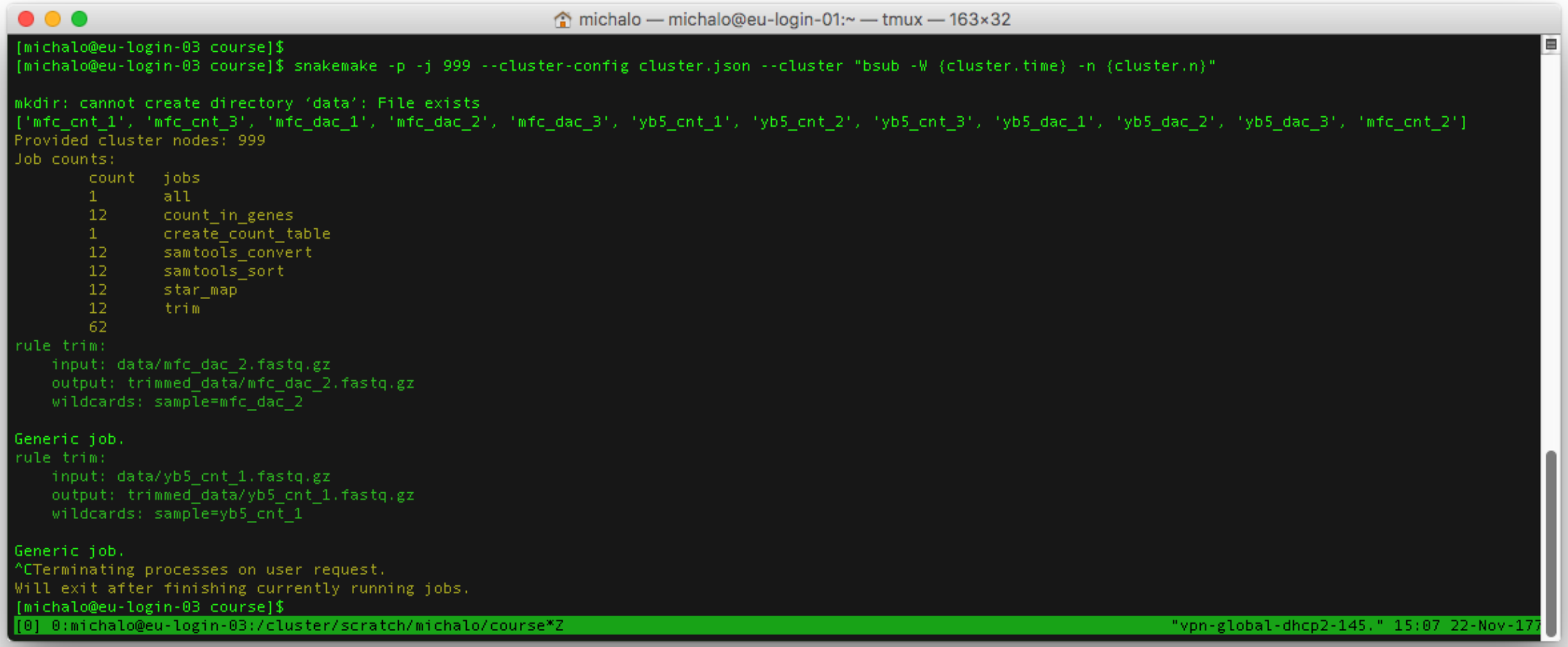
```
rule all:
    input:
        expand("edgeR/02_analyze_DE/{de_subset}.P1e-3_C{log2FC_cutoff}.DE.annotated.sorted.xlsx",
               de_subset=de_subset,
               log2FC_cutoff=config['log2FC_cutoff'])
```

Then all rules needed to produce the input(s) of ***rule all*** are confirmed/verified from **bottom-to-top**

Snakefile – rules with python

```
rule create_count_table:
    input:
        expand("secondary_analysis/{sample}.cnt", sample = SAMPLES)
    output:
        "secondary_analysis/counts.csv"
    run:
        import pandas
        import glob

        filez = glob.glob('secondary_analysis/*.cnt')
        t1 = pandas.read_table(filez[1], header=1)
        tout = t1.iloc[:,0]
        for f in filez:
           t1=pandas.read_table(f, header=1)
           tout=pandas.concat([tout, t1.iloc[:,6]], axis=1)
           print(f)

        tout.to_csv('secondary_analysis/counts.csv')
```

Running snakemake on the cluster



Snakemake vs Nextflow

- Δεν υπάρχει ξεκάθαρος νικητής.
- Το Snakemake και το Nextflow είναι και τα δύο ανοιχτού κώδικα, δημιουργήθηκαν περίπου ταυτόχρονα και προσφέρουν παρόμοιες λύσεις στη διαχείριση ροής εργασιών.
- Και οι δύο έχουν ευδοκιμήσει από τότε με μια μεγάλη κοινότητα χρηστών και στις δύο πλευρές.

Snakemake vs Nextflow

- **Γλώσσα**: Το Snakemake βασίζεται στην Python, η οποία μπορεί να είναι πιο οικεία σε μερικούς, ενώ το Nextflow βασίζεται στο Groovy, κάτι που μπορεί να κάνει πιο δύσκολο και μεγαλύτερο τον έλεγχο, ανάλογα με το υπόβαθρό σας.

Snakemake vs Nextflow

- **Προσέγγιση:** Το Nextflow χρησιμοποιεί μια προσέγγιση «από πάνω προς τα κάτω» που σέβεται τη φυσική ροή της ανάλυσης δεδομένων,
- το Snakemake χρησιμοποιεί μια προσέγγιση «από κάτω προς τα πάνω», η οποία συνεπάγεται την προεκτίμηση όλων των εξαρτήσεων, ξεκινώντας από τα αναμενόμενα αποτελέσματα και λειτουργώντας προς τα πίσω στο ακατέργαστα δεδομένα. Αυτό, προφανώς, θα μπορούσε να είναι ένας περιοριστικός παράγοντας για πολύ μεγάλους υπολογισμούς και ροές εργασίας (μόνο για ακραίες περιπτώσεις).

Snakemake vs Nextflow

- **Ορισμός εξόδου:** Στο Snakemake, οι χρήστες πρέπει να ορίσουν ρητά τα ονόματα και τους φακέλους των αρχείων εξόδου (τα οποία, ακόμα κι αν είναι κουραστικό, μπορούν να κάνουν μια διοχέτευση πιο κατανοητή για ορισμένους). Αυτό συμβαίνει επειδή βασίζεται σε αυτά τα αρχεία για να προσδιορίσει πότε θα εκτελεστεί κάθε βήμα του αγωγού. Έχοντας μια προσέγγιση "από πάνω προς τα κάτω", το Nextflow δεν χρειάζεται τα αποτελέσματα να ονομάζονται ρητά.

Snakemake vs Nextflow

- **Dry runs:** Οι ξηρές εκτελέσεις είναι χρήσιμες για τη δοκιμή ροών εργασιών, στο Snakemake, αυτό μπορεί να γίνει χωρίς δεδομένα, ενώ στο Nextflow, πρέπει να παρέχετε τουλάχιστον ένα μικρό σύνολο δεδομένων (αλλά αυτή είναι πιθανώς μια καλή ιδέα ούτως ή άλλως για να ελέγξετε για περαιτέρω σφάλματα) .

Snakemake vs Nextflow

- **Κατάλογος δημοσιευμένων και επιμελημένων pipelines:** Το έργο κοινότητας χρηστών του Nextflow (nf-core) είναι περισσότερο γνωστό, αλλά ένας ισοδύναμος και λιγότερο γνωστός κατάλογος υπάρχει επίσης για το Snakemake (ροές εργασιών Snakemake).

Wratten, L., Wilm, A. & Göke, J. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nat Methods* **18**, 1161–1168 (2021). doi: 10.1038/s41592-021-01254-9

snakemake

```
rule count:
    input: bams = expand('{sample}.Aligned.sortedByCoord.out.bam', sample=samples)
    threads: config['featureCounts_threads']
    output: counts='counts.txt'
    conda: "envs/rnaseq.yaml"
    params: gtf=config['gtf']
    shell:
        """ featurecounts -M
                          -s 0
                          -T {threads}
                          -p
                          -t exon
                          -g gene_id
                          -a {params.gtf}
                          -o {output.counts} {input.bams} """
```

nextflow

```
bams = Channel.fromPath("*.Aligned.sortedByCoord.out.bam")

// Process 'count'
process count:

    conda '/envs/rnaseq.yaml'

    input:
        set file(bam) from bams.collect()

    output:
        counts='counts.txt'

    shell:

       """
        featurecounts -M \
                      -s 0 \
                      -T ${params.featureCounts_threads} \
                      -p \
                      -t exon \
                      -g gene_id \
                      -a ${params.gtf} \
                      -o ${output.counts} \
                      ${input.bams}
        """
```

Workflow logic is the most important skill to develop

snakemake

```
rule count:
    input: bams = expand('{sample}.Aligned.sortedByCoord.out.bam', sample=samples)
    threads: config['featureCounts_threads']
    output: counts='counts.txt'
    conda: "envs/rnaseq.yaml"
    params: gtf=config['gtf']
    shell:
        """ featurecounts -M
                          -s 0
                          -T {threads}
                          -p
                          -t exon
                          -g gene_id
                          -a {params.gtf}
                          -o {output.counts} {input.bams} """
```

nextflow

```
bams = Channel.fromPath("*.Aligned.sortedByCoord.out.bam")

// Process 'count'
process count:

    conda '/envs/rnaseq.yaml'

    input:
        set file(bam) from bams.collect()

    output:
        counts='counts.txt'

    shell:

       """
        featurecounts -M \
                      -s 0 \
                      -T ${params.featureCounts_threads} \
                      -p \
                      -t exon \
                      -g gene_id \
                      -a ${params.gtf} \
                      -o ${output.counts} \
                      ${input.bams}
        """
```

nextflow offers more flexibility - 1

Multi-language support

```
process perlStuff {

    """
    #!/usr/bin/perl

    print 'Hi there!' . '\n';
    """

}

process pyStuff {

    """
    #!/usr/bin/python

    x = 'Hello'
    y = 'world!'
    print "%s - %s" % (x,y)
    """

}
```

Yep, this is Python2!

Flexibility in process definition

```
seq_to_align = ...
mode = 'tcoffee'

process align {
    input:
    file seq_to_aln from sequences

    script:
    if( mode == 'tcoffee' )
        """
        t_coffee -in $seq_to_aln > out_file
        """

    else if( mode == 'mafft' )
        """
        mafft --anysymbol --parttree --quiet $seq_to_aln > out_file
        """

    else if( mode == 'clustalo' )
        """
        clustalo -i $seq_to_aln -o out_file
        """

    else
        error "Invalid alignment mode: ${mode}"

}
```

nextflow offers more flexibility - 2

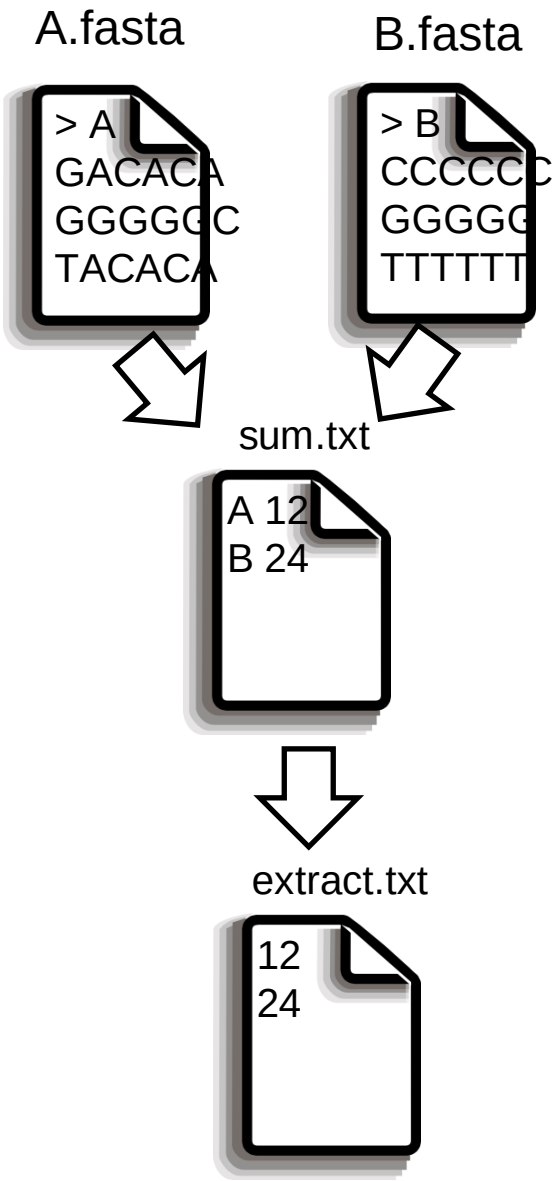
- Abstraction between **pipeline processes** and **how the pipeline will be executed**

- 'Recycle' processes between different workflows

```
process foo {
    output:
    path 'foo.txt'

    script:
    """
    your_command > foo.txt
    """
}

process bar {
    input:
    path x

    output:
    path 'bar.txt'

    script:
    """
    another_command $x > bar.txt
    """
}

workflow {
    data = channel.fromPath('/some/path/*.txt')
    foo()
    bar(data)
}
```

Practice

First example



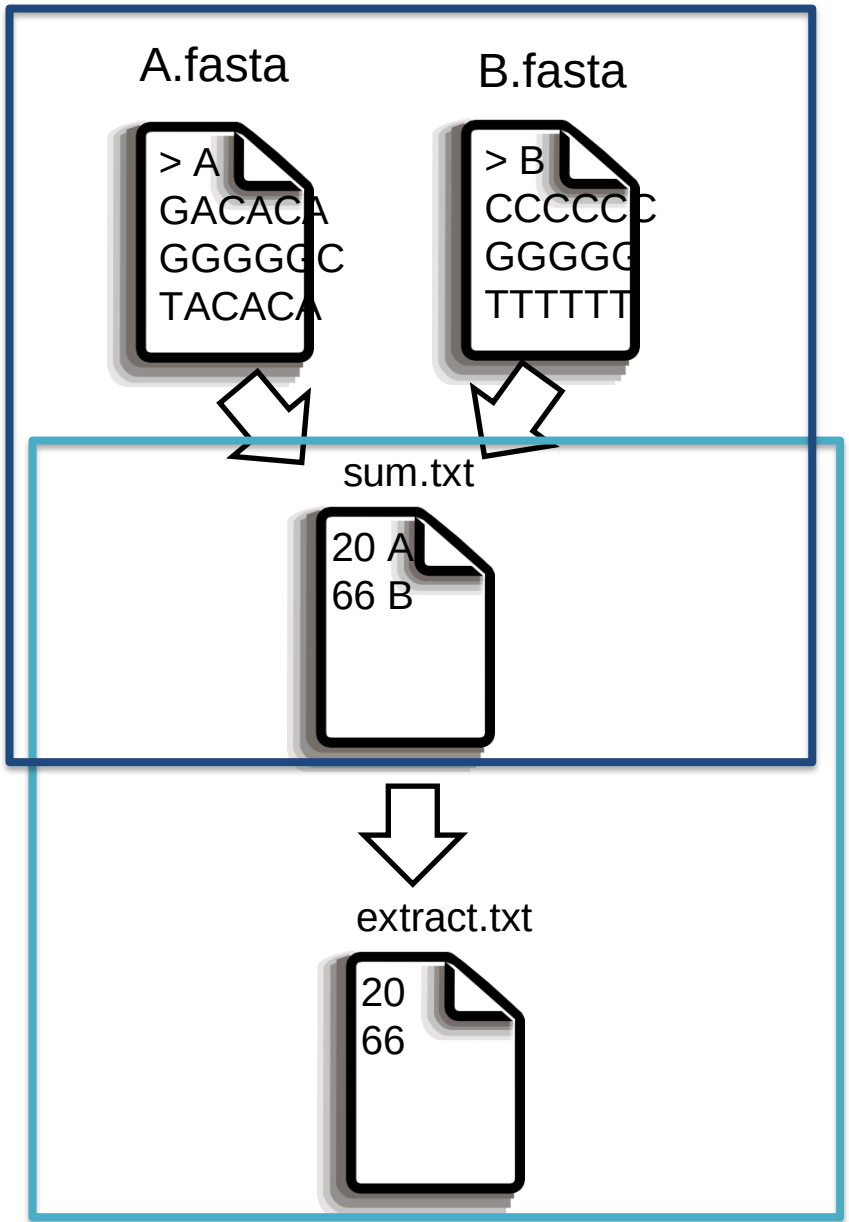
Input files

A.fasta
B.fasta

snakemake file

Snakefile

First rules



rule summarize:

input:

“A.fasta”,

“B.fasta”

output:

“sum.txt”

shell:

“wc –c {input} > {output}”

rule extract:

input:

“sum.txt”

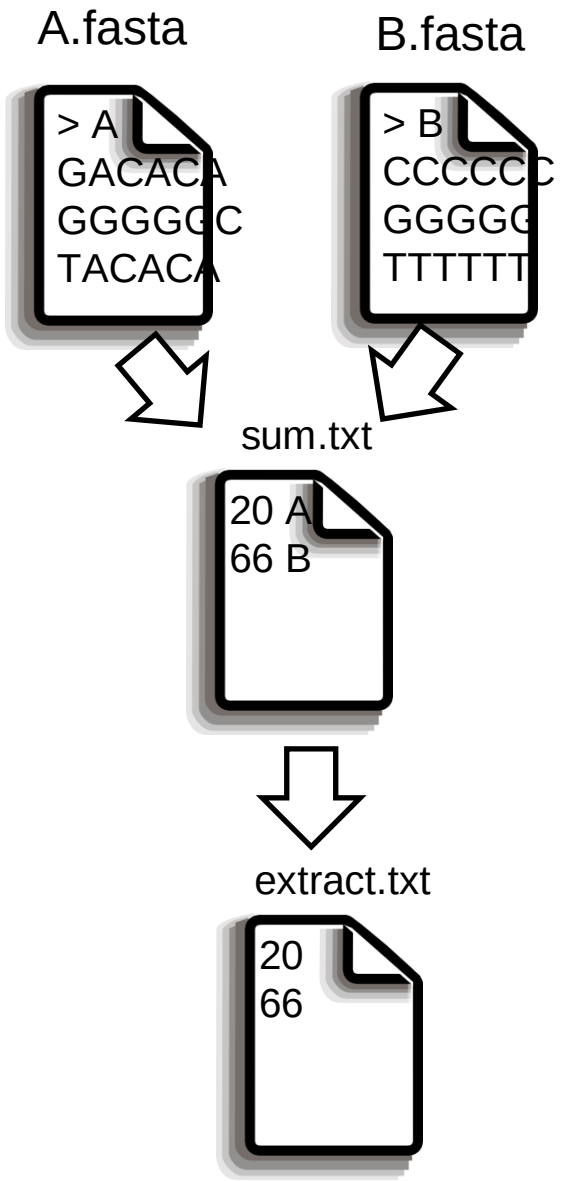
output:

“extract.txt”

shell:

“cut -f1 –d ‘ ‘ {input} > {output}”

Run the rules on its own

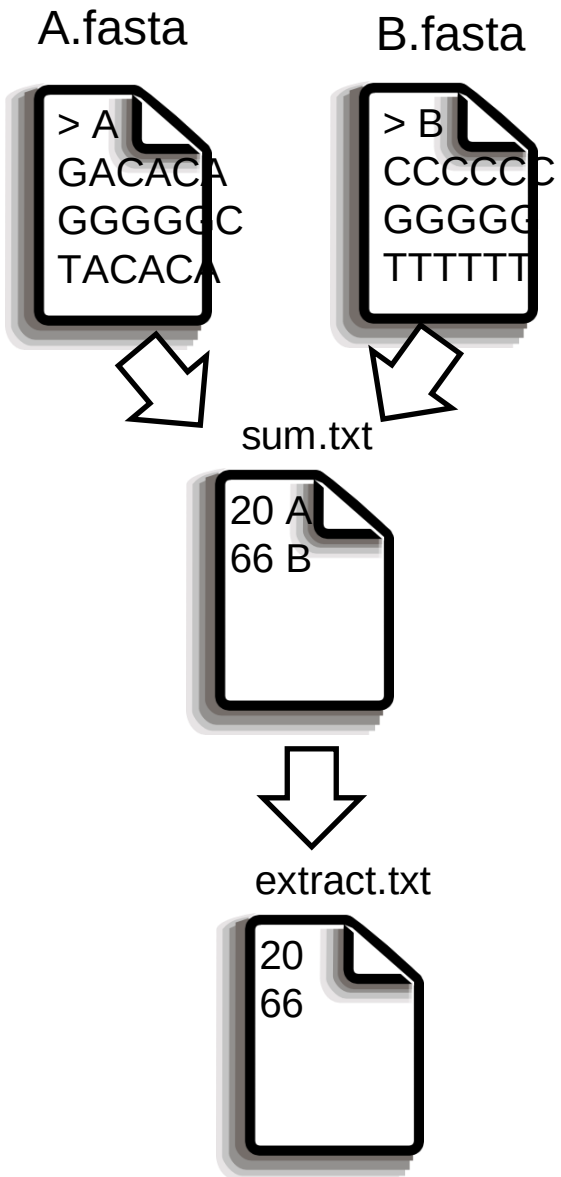


> snakemake summarize

> snakemake extract

> rm sum.txt extract.txt

Add the default rule



Add the rule to the top of the Snakefile

```
rule all:
        input:
                "extract.txt"
```

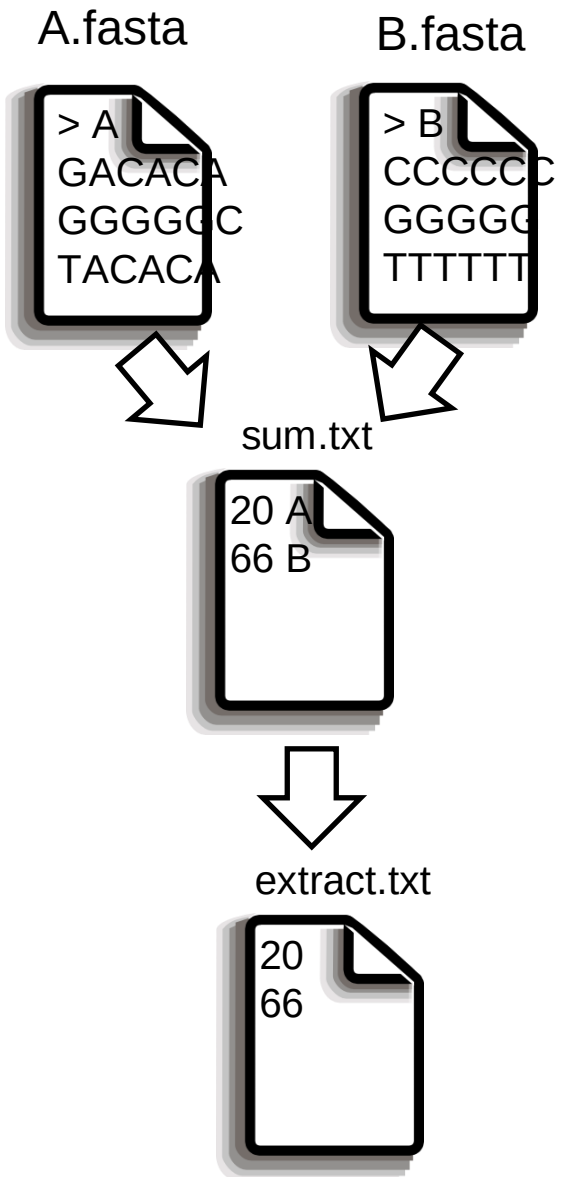
simply run

> **snakemake**

cleaning up

> **rm sum.txt extract.txt**

Add a clean rule



Snakefile

```
rule clean:
        shell:
                "rm –f extract.txt sum.txt"
```

then run

```
> snakemake clean
```

Execute

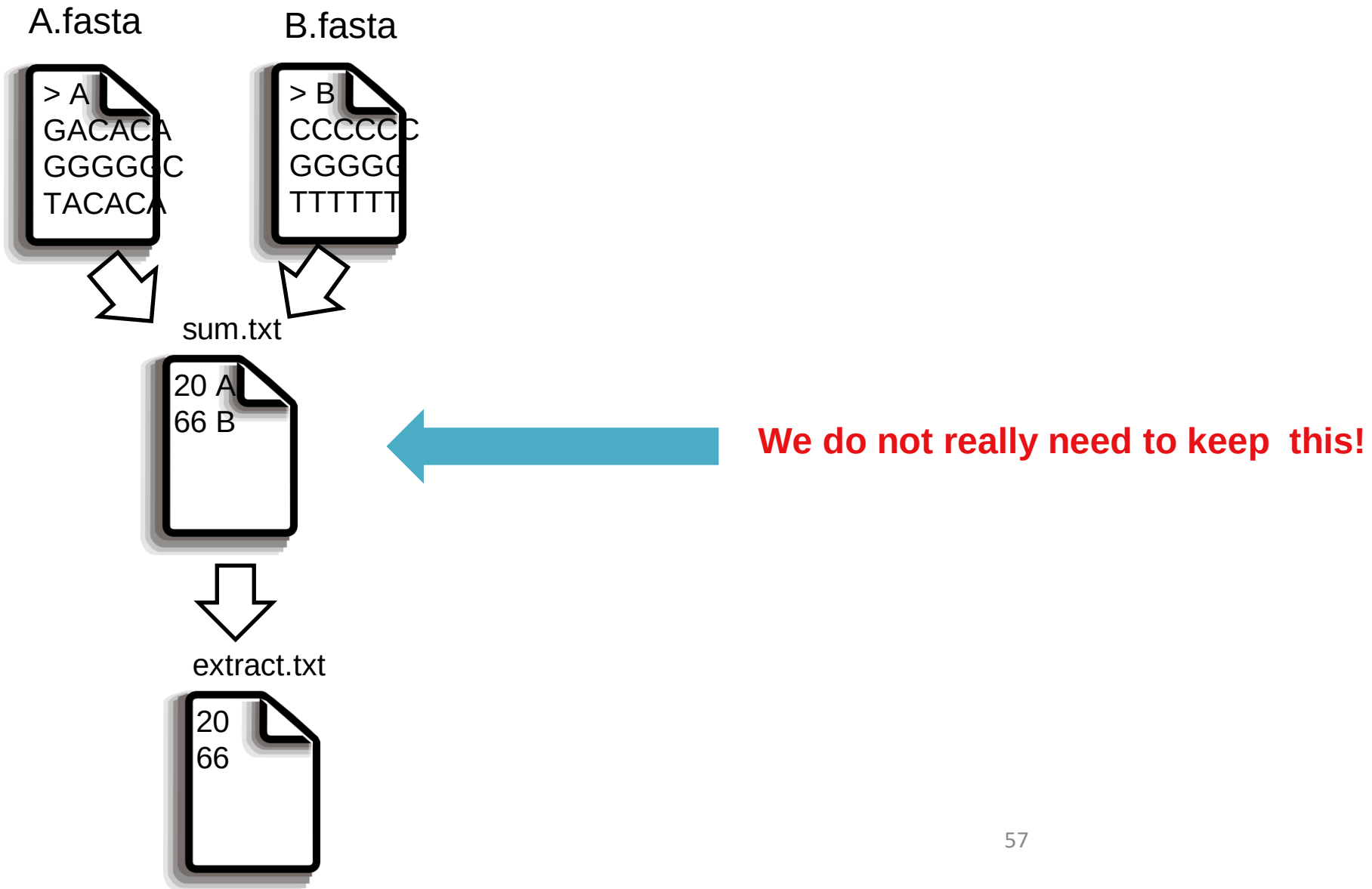
```
> snakemake          # executes rule 'all'

> snakemake –n       # dry run

> snakemake –p       # print out the commands
```

Add a message

Snakefile

```
rule summarize:
    input:
        "A.fasta",
        "B.fasta"
    output:
        "extract.txt"

    message:
        "summarizing counts from {input} and creating {output}"

    shell:
        "wc –c {input} > {output}"
```

Add a temporary variable



Add a temporary

Snakefile

```
SAMPLES = "A B".split()

rule summarize:
        input:
                expand("{samples}.fasta", samples=SAMPLES)

        output:
                temp("sum.txt")

        message:
                "extracting counts from {input} and creating {output}"
        shell:
                "wc –c {input} > {output}"
```

> **snakemake clean**

> **snakemake**

Add a log file

Snakefile

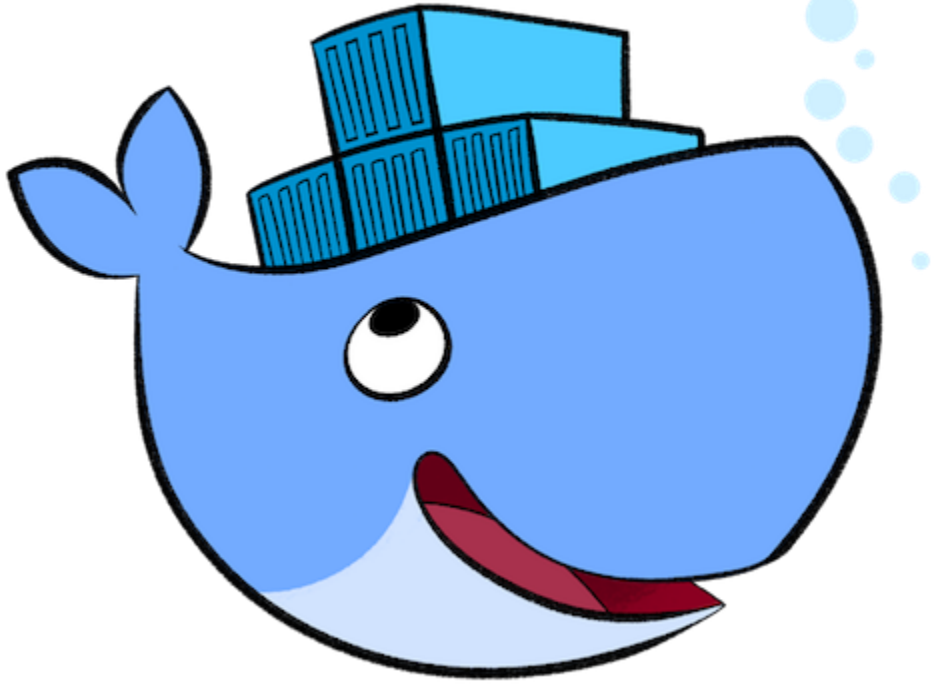
```
rule summarize:
        input:
                "{samples}.fasta"
        output:
                "sum.txt"
        message:
                "extracting counts from {input} and creating {output}"
        log:
                "sum.log"
        shell:
                "wc –c {input} > {output} 2> {log}"
```

```
> snakemake clean
> snakemake
```


docker

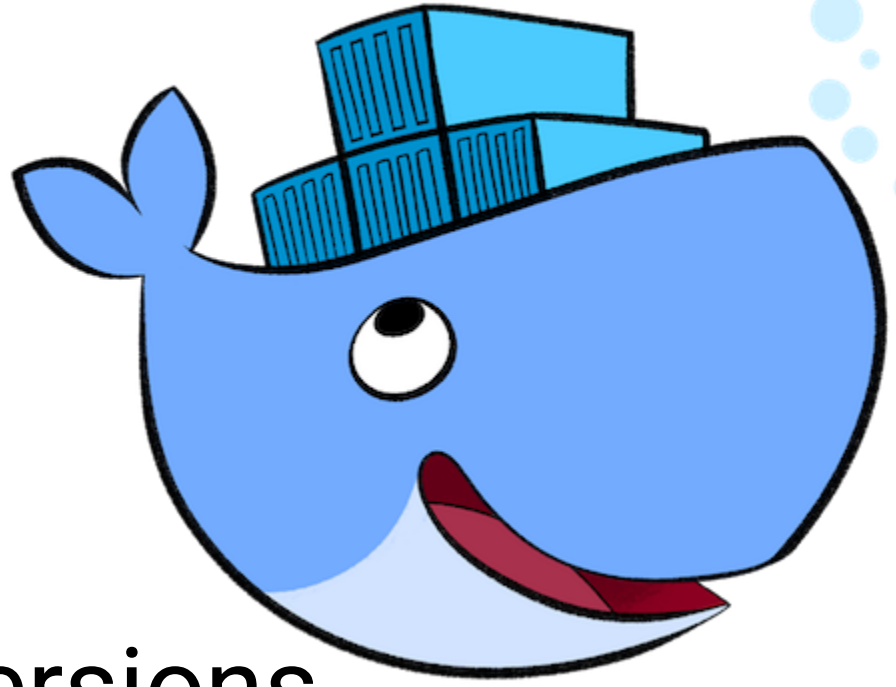
Why do we need containers?

1. ‘Integration hell’



2. ‘It works on my laptop’ problem

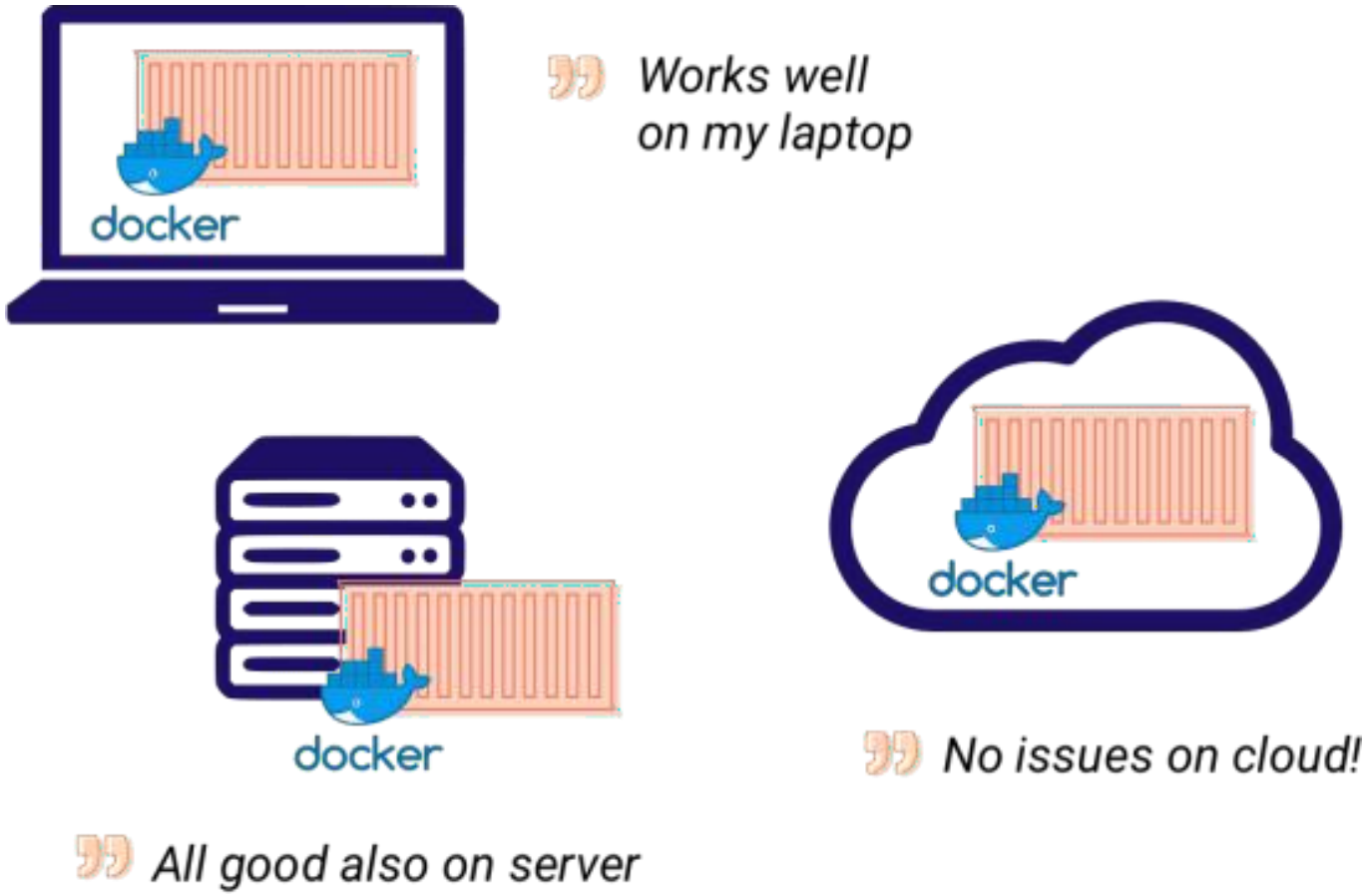
Deployment issues of bioinformatics tools



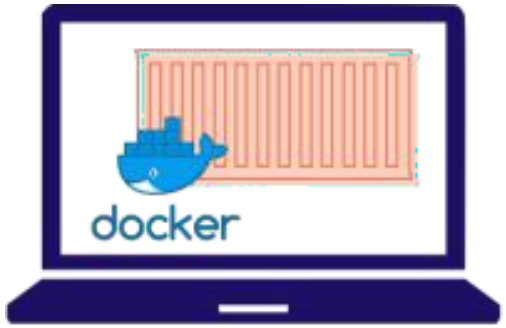
- Different environments
- Different OS
- Different packaging
- Conflict between tools or versions

- Impact on usability and reproducibility

Containerized applications run **the same** across different computing platforms

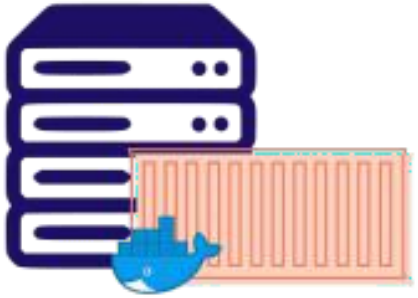


Containerized applications run **the same** across different computing platforms



Works well on my laptop

Interoperability



All good also on server

No issues on cloud!

Different levels of encapsulation

Goal : capture the system environment of applications (OS, packages, libraries,. . .) to control their execution.

- Hardware virtualisation (virtual machines) 
- OS virtualisation (images and containers) 

- Environment management CONDA

Encapsulation

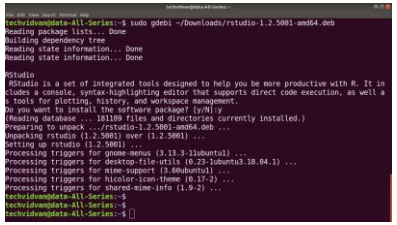
Ας υποθέσουμε ότι θέλουμε να εγκαταστήσουμε το RStudio...

Install Rstudio ?

MacOS

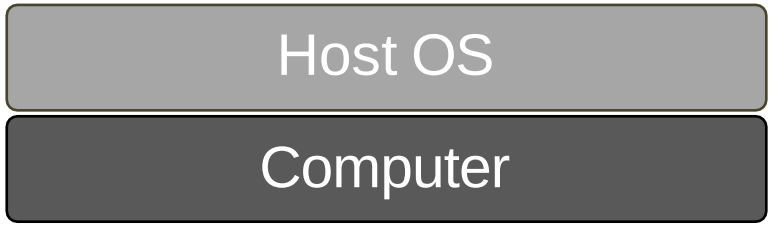
Windows

Unix-based



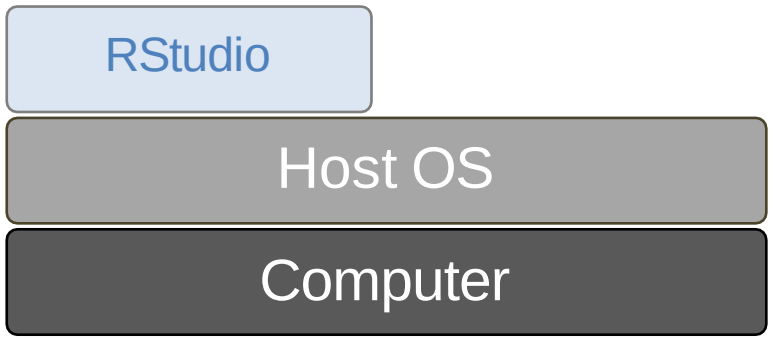
Use Rstudio

Encapsulation



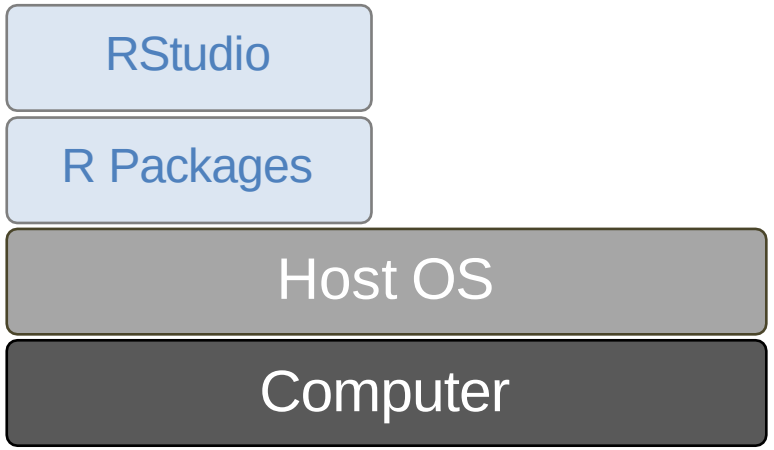
Ξεκινήσαμε με έναν υπολογιστή που χρησιμοποιεί ένα συγκεκριμένο λειτουργικό σύστημα...

Encapsulation



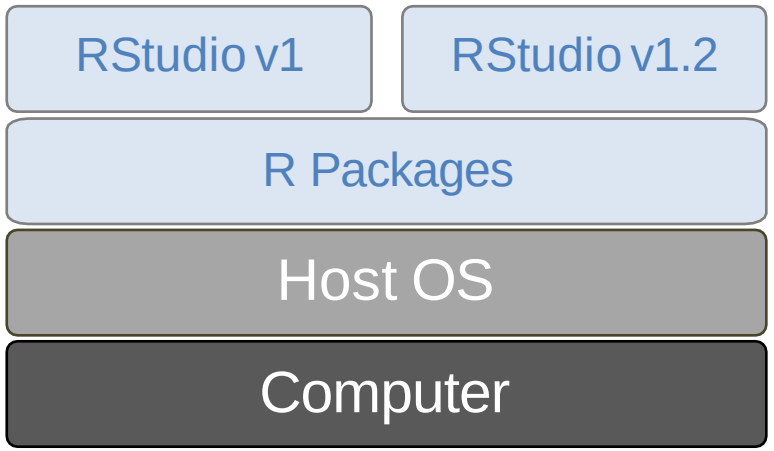
We started with a computer using a specific OS...
And inside this environment, we installed a new application.

Encapsulation



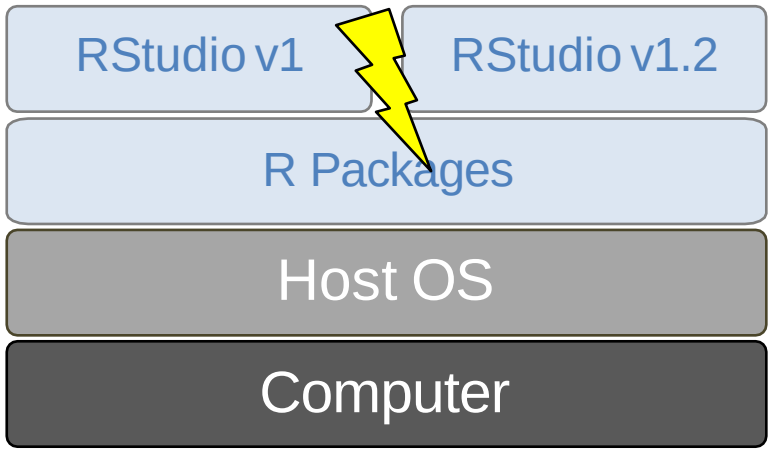
Ξεκινήσαμε με έναν υπολογιστή που χρησιμοποιεί ένα **συγκεκριμένο λειτουργικό σύστημα**...
Και μέσα σε αυτό το περιβάλλον, εγκαταστήσαμε μια νέα εφαρμογή.
Οι εφαρμογές βασίζονται σε εξαρτήσεις,
π.χ. εξωτερικές βιβλιοθήκες.

Encapsulation



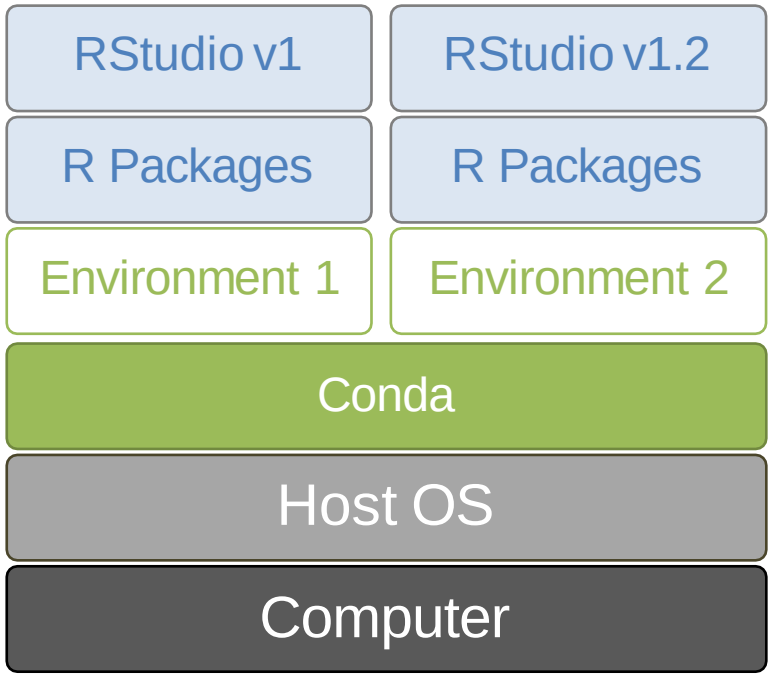
- Συνήθως οι εξαρτήσεις διαφορετικών εφαρμογών δεν παρεμβαίνουν.
- Τι γίνεται όμως αν θέλουμε να δοκιμάσουμε την πιο πρόσφατη έκδοση του αγαπημένου μας εργαλείου;
- Μπορεί να υπάρχουν συγκρούσεις. . .

Encapsulation



- Συνήθως οι εξαρτήσεις διαφορετικών εφαρμογών δεν παρεμβαίνουν.
- Τι γίνεται όμως αν θέλουμε να δοκιμάσουμε την πιο πρόσφατη έκδοση του αγαπημένου μας εργαλείου;
- Μπορεί να υπάρχουν συγκρούσεις. . .

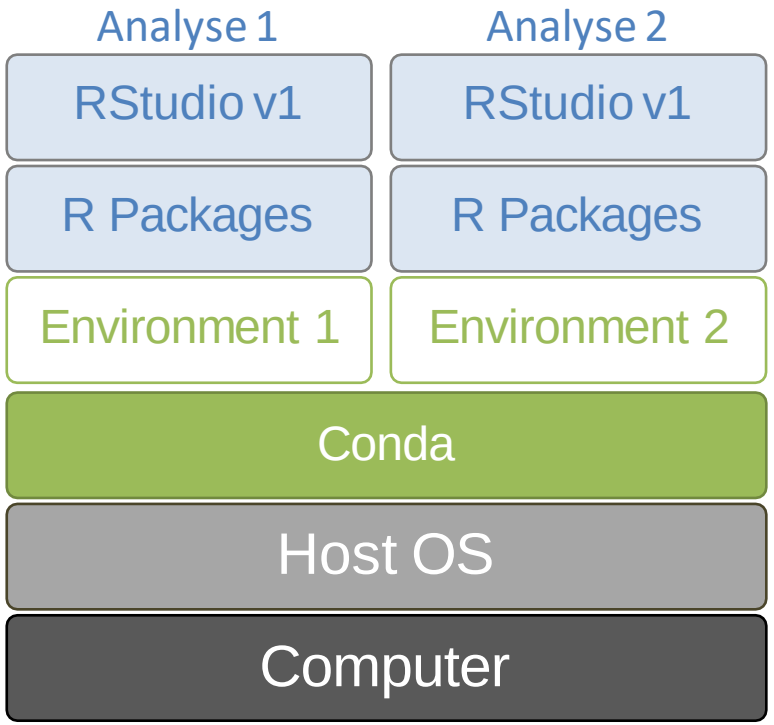
Encapsulation : managing environments



Ιδέα : δημιουργία ξεχωριστών περιβαλλόντων για κάθε εφαρμογή.

CONDA

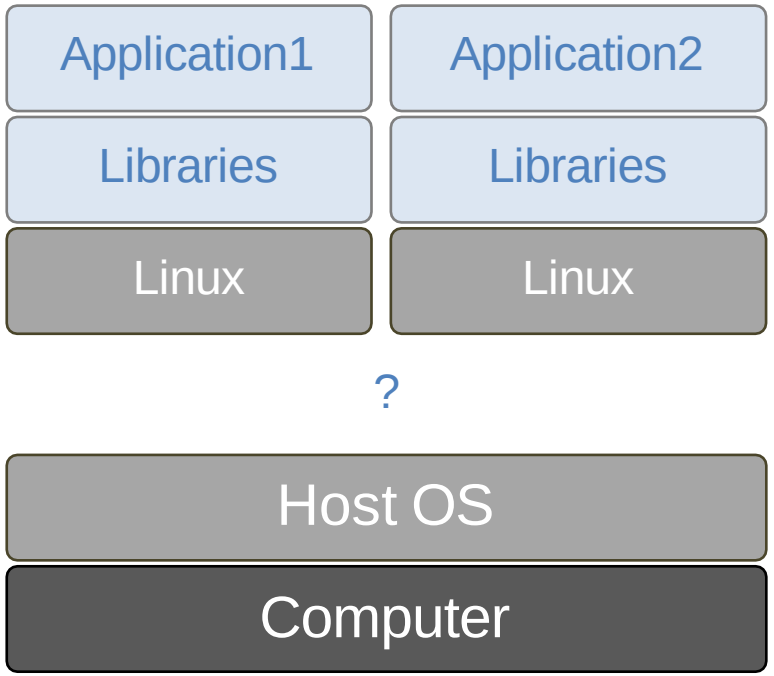
Encapsulation : managing environments



Ιδέα : δημιουργία ξεχωριστών περιβαλλόντων για κάθε εφαρμογή.

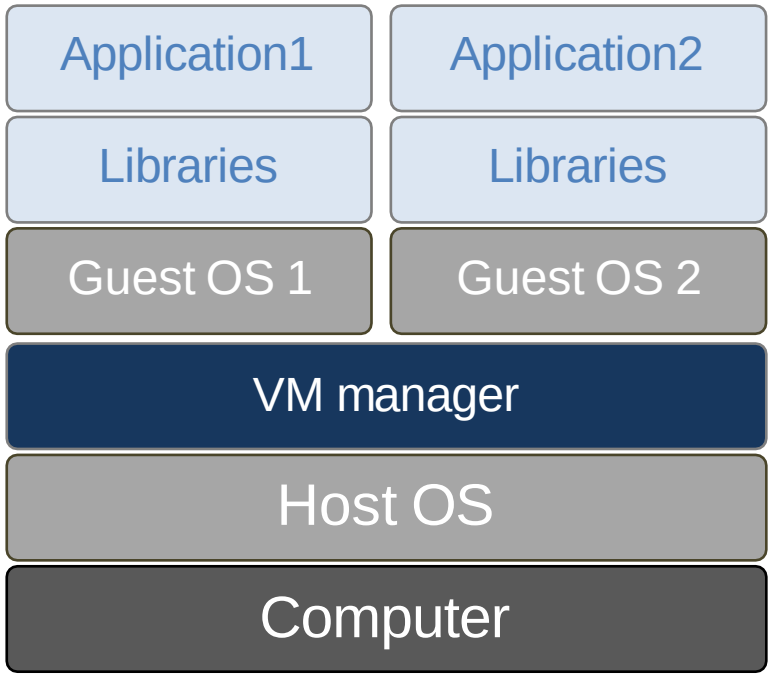
Πιο ευέλικτο: δημιουργήστε ένα νέο περιβάλλον ανά ανάλυση.

Encapsulation : hardware virtualisation



Τι γίνεται όμως αν θέλουμε να εγκαταστήσουμε ένα λογισμικό από διαφορετικό λειτουργικό σύστημα;

Encapsulation : hardware virtualisation



Ιδέα: χρησιμοποιήστε εικονικές μηχανές

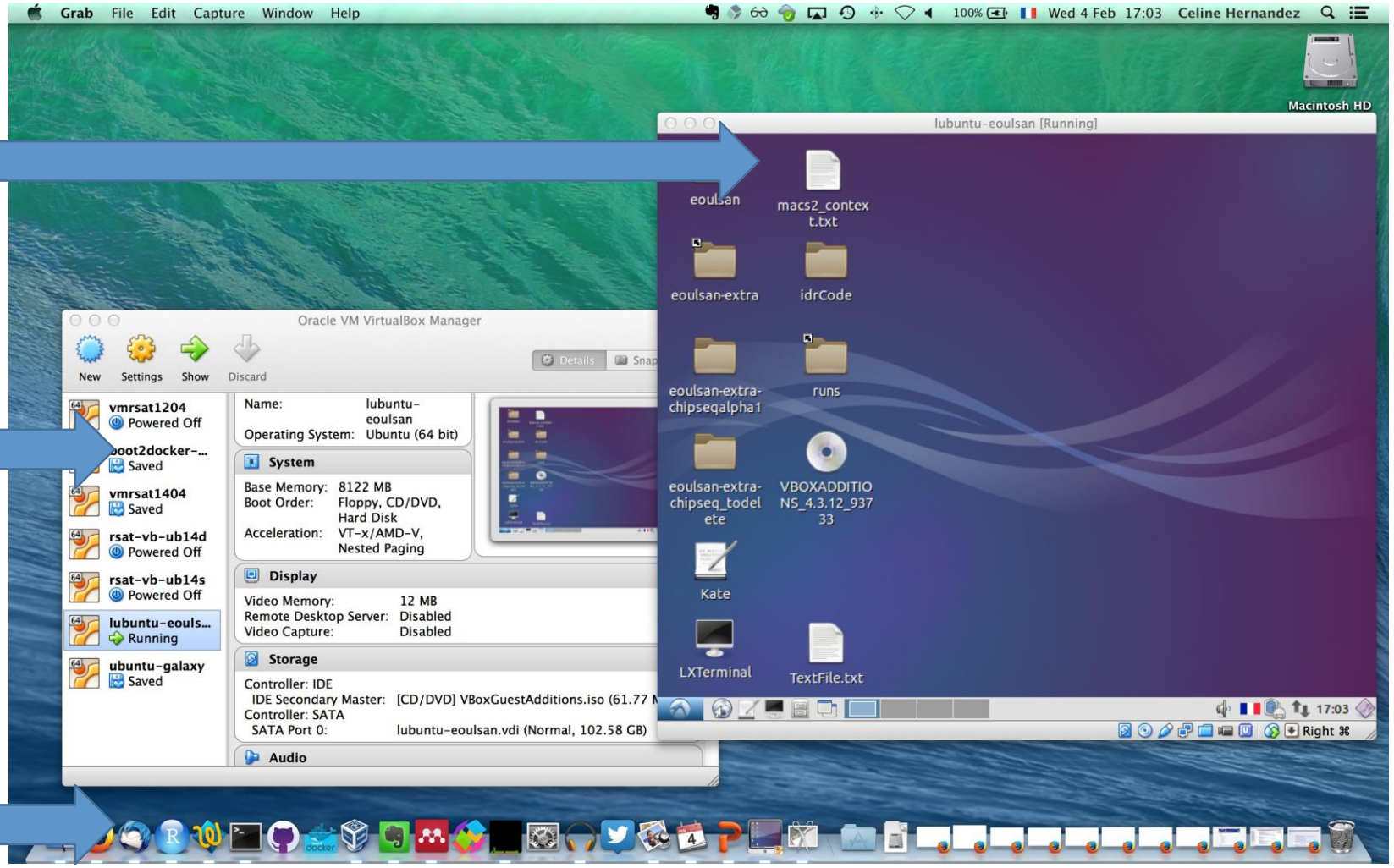
Πλεονεκτήματα:

- Κάθε εφαρμογή αποκτά ένα εντελώς διαφορετικό και ανεξάρτητο περιβάλλον
- Οι εικονικές μηχανές μπορούν να μεταφερθούν σε άλλον υπολογιστή (χρησιμοποιώντας τον ίδιο διαχειριστή)

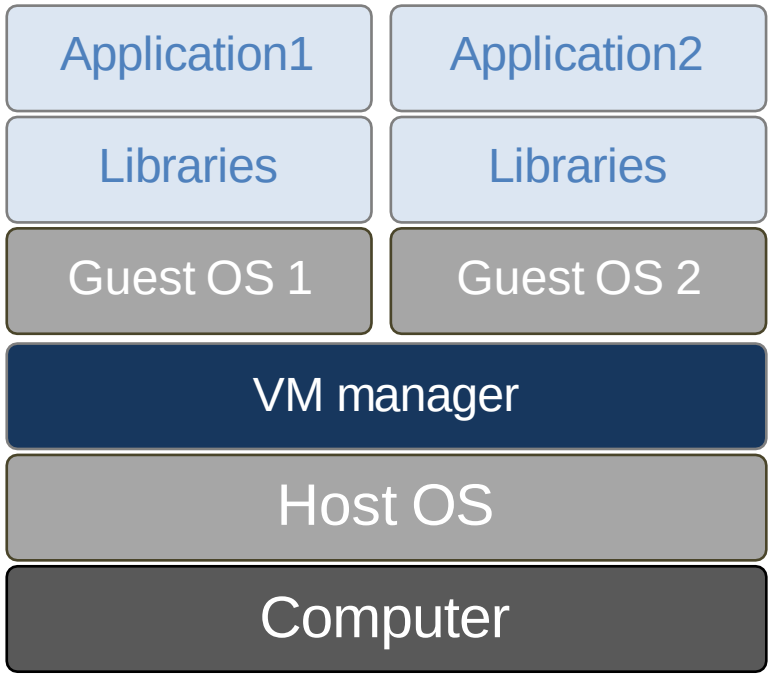
Encapsulation : hardware virtualisation

Ubuntu

MacOS



Encapsulation : hardware virtualisation



Ιδέα: χρήση εικονικών μηχανών

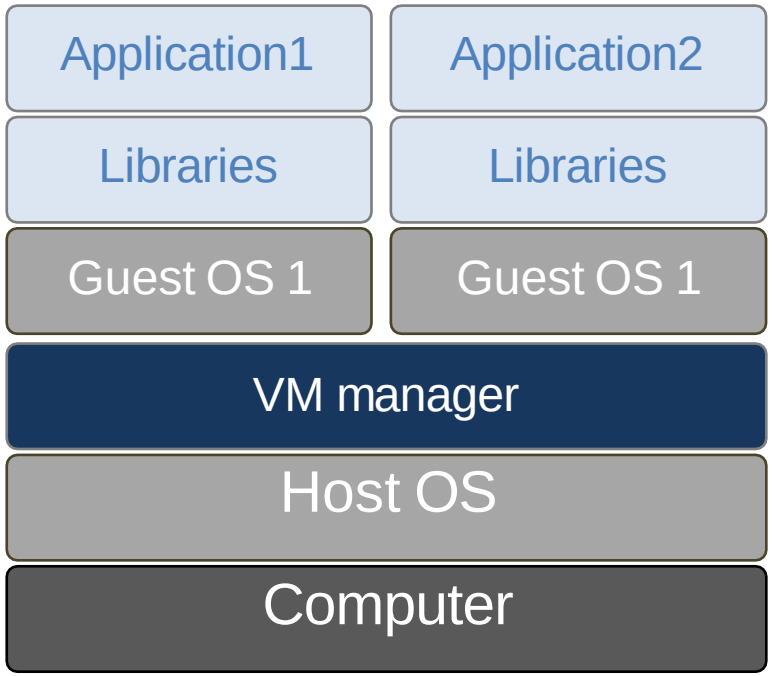
Πλεονεκτήματα:

- μεταφερόμενα ανεξάρτητα περιβάλλοντα

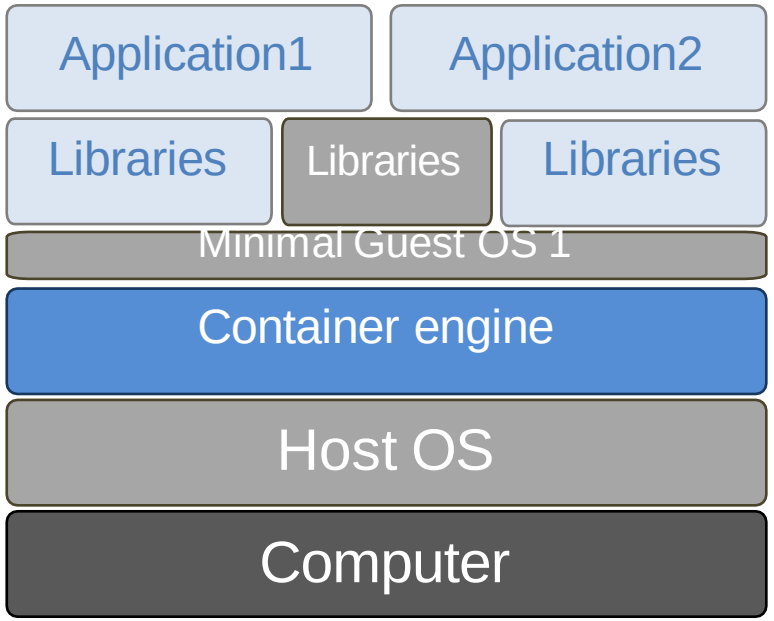
Μειονεκτήματα:

- Πλεονασμός μεταξύ VMs
- Heavy για ρύθμιση
- Χωρίς αυτοματισμό

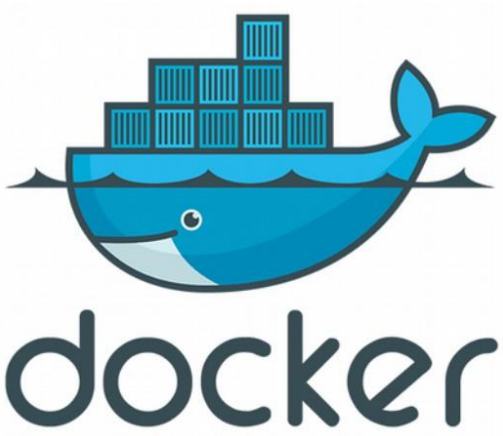
Encapsulation : OS virtualisation



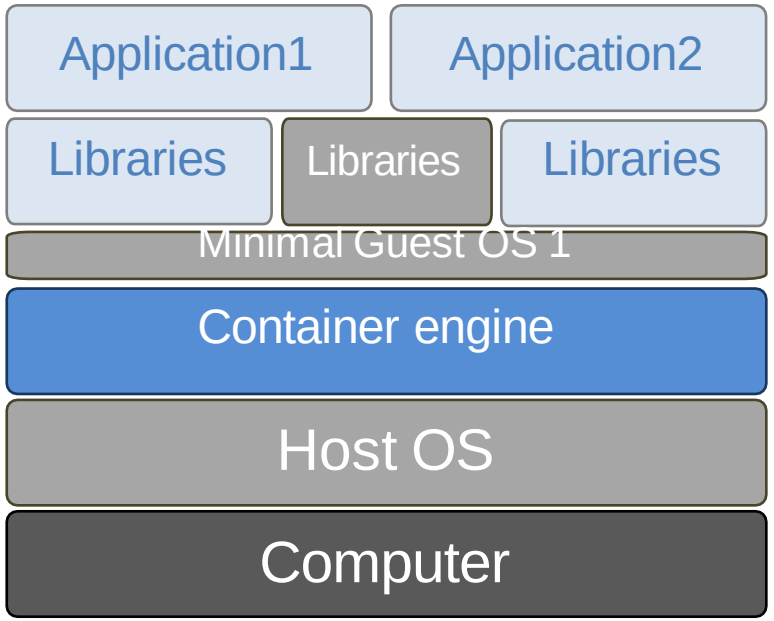
Encapsulation : OS virtualisation



Ιδέα: «ξεγελάστε» τις εφαρμογές για να πιστέψουν ότι βρίσκονται σε διαφορετικό λειτουργικό σύστημα από αυτό του κεντρικού υπολογιστή
Αποφύγετε τον πλεονασμό.



Encapsulation : OS virtualisation

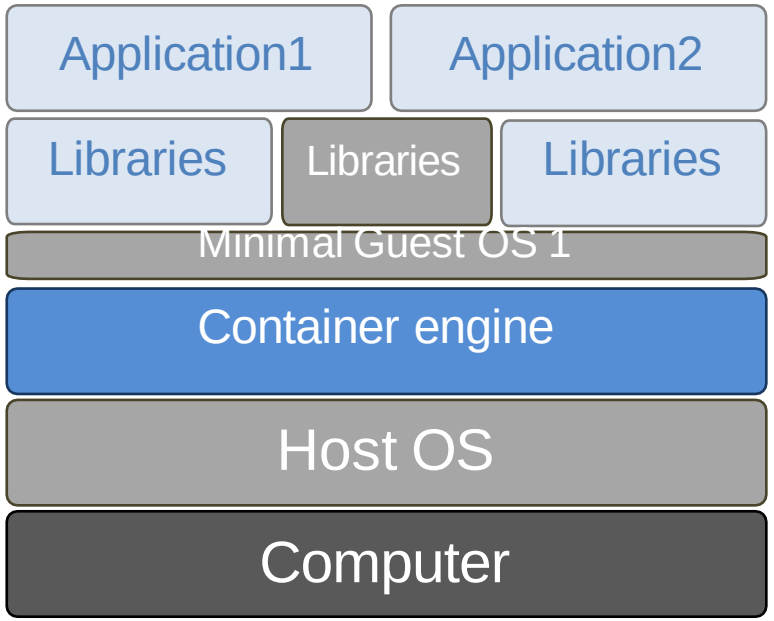


OS virtualisation vs hardware virtualisation

Πλεονεκτήματα:

- Ταχύτητα
  - Η εγκατάσταση είναι πιο γρήγορη
  - Δεν υπάρχει χρόνος εκκίνησης
- Lightweight
  - Minimal Base OS
  - Ελάχιστες βιβλιοθήκες και σετ εφαρμογώ
- Εύκολη κοινή χρήση εφαρμογών

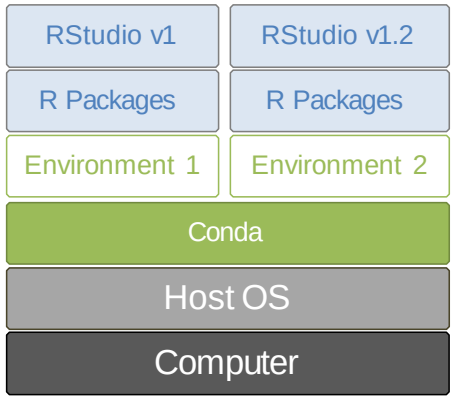
Encapsulation : OS virtualisation

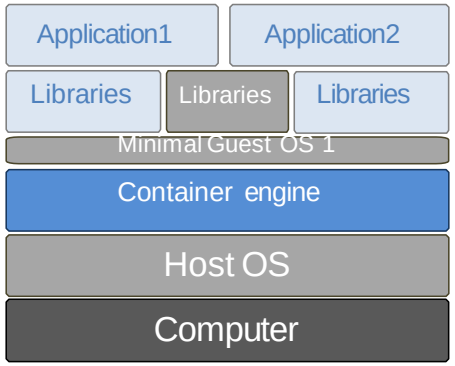


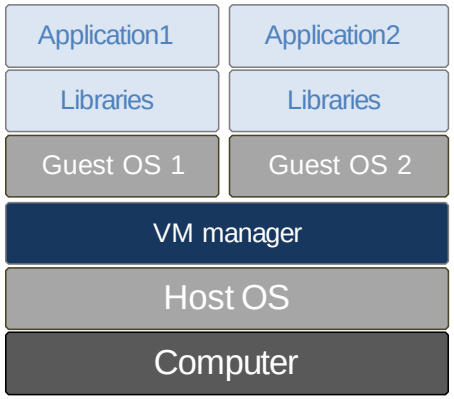
Μειονεκτήματα:

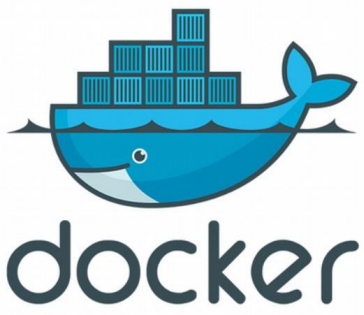
- Μοναδικότητα στη χρήση εικόνων σε ένα σύμπλεγμα
- Αλλαγές πολιτικών της εταιρείας Docker

Encapsulation

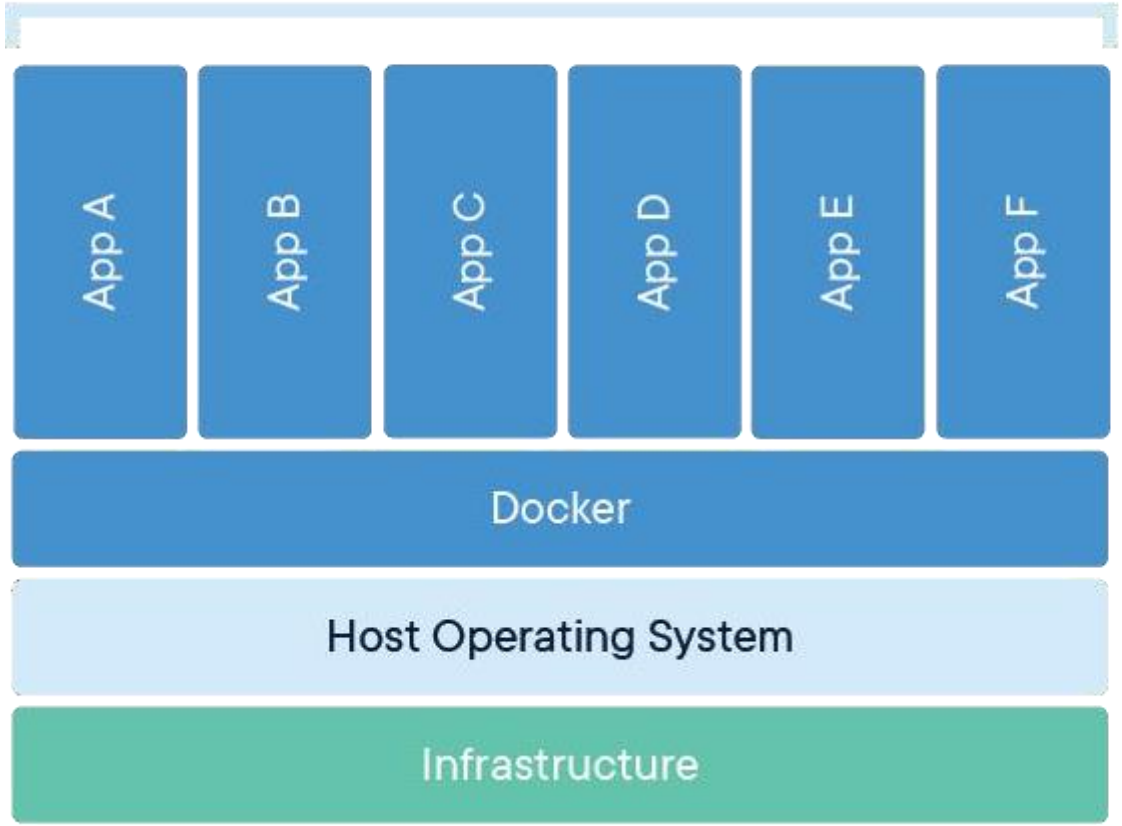




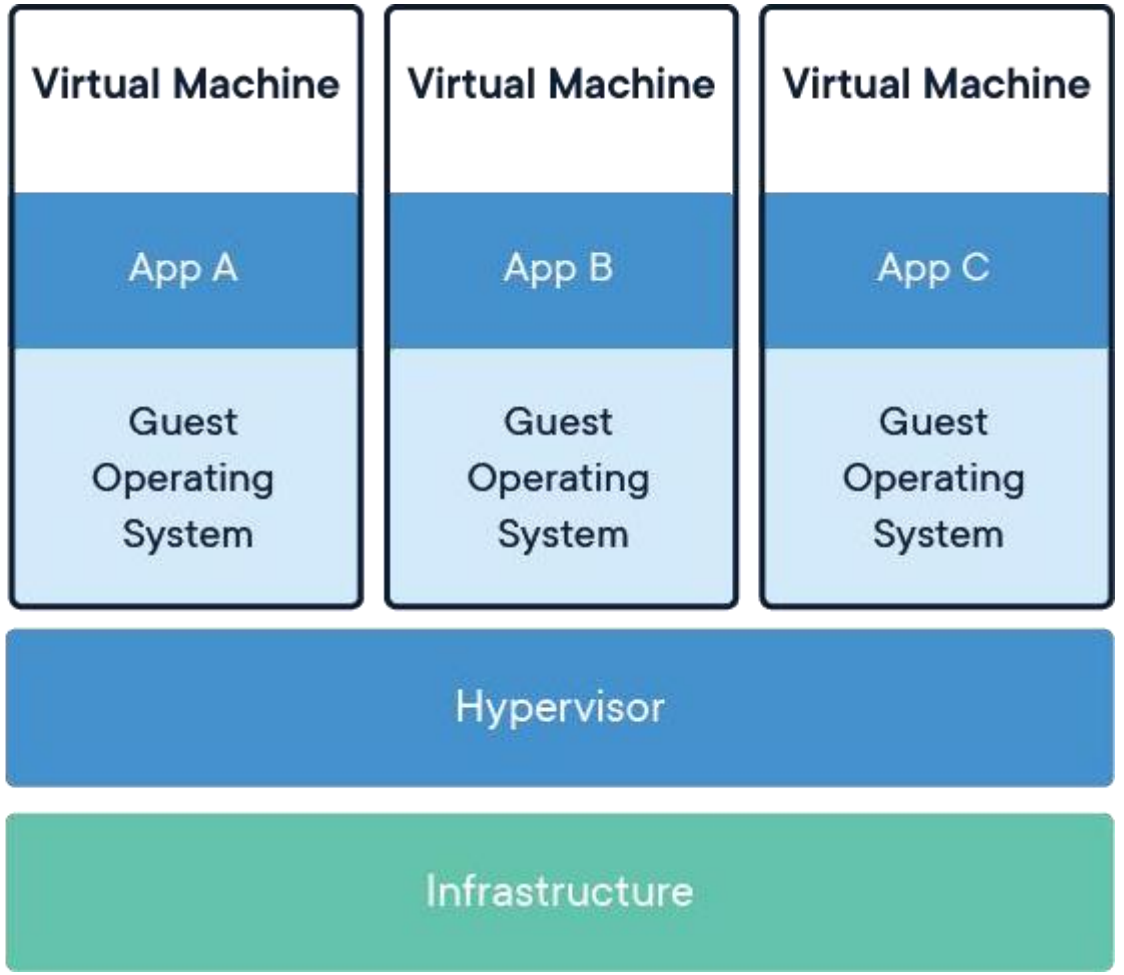




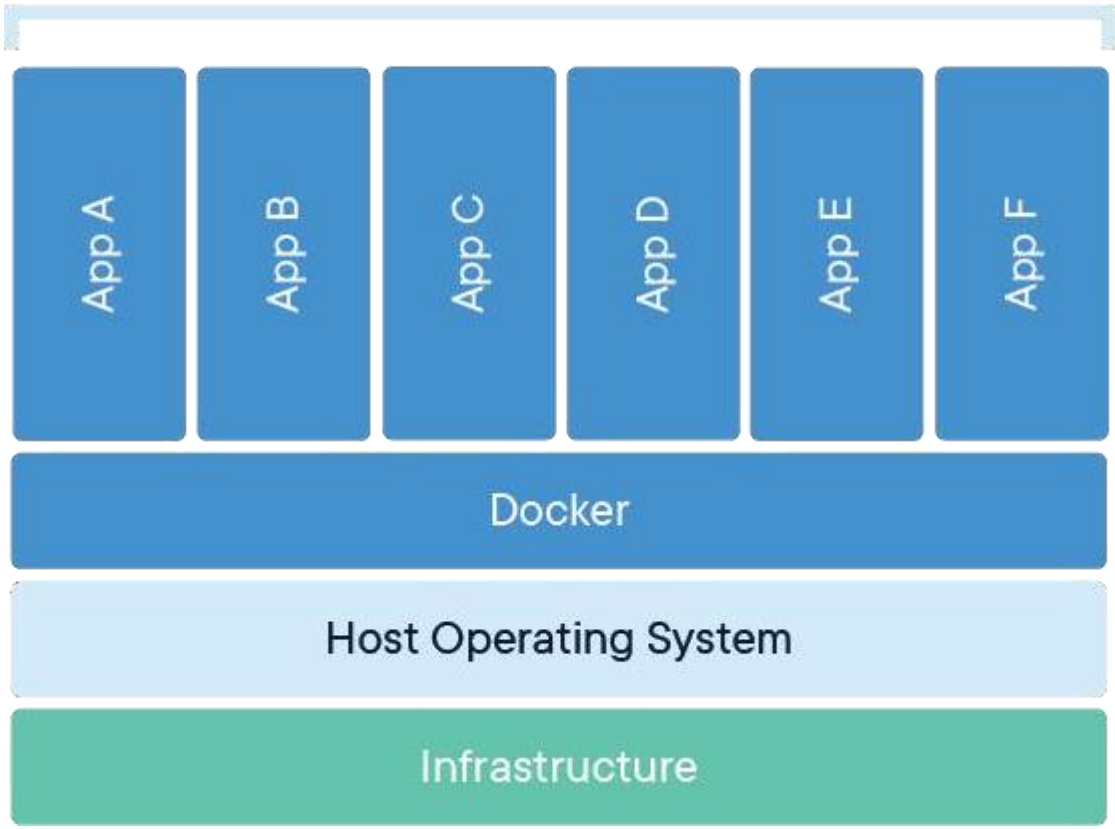
Containerized Applications

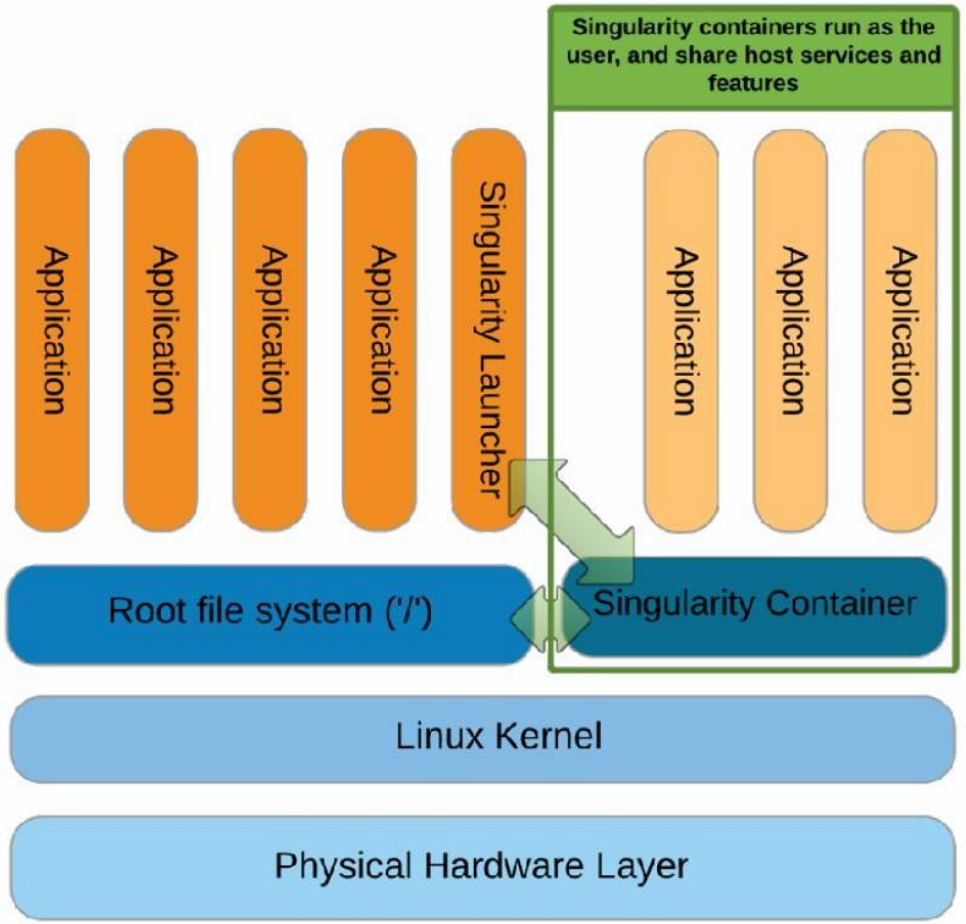


Virtual Machines (VMs)



Containerized Applications






CONDA

CONDA : an environment manager

Conda definitions

- Environment: a set of packages/tools in a directory (added to our PATH)
- Conda: an open source package + a general-purpose environment management system (installation, execution, upgrade). For any programming language, multi-platform (Windows, MacOS, Linux).
- Conda package: a compressed tarball of a tool

Why using an environment manager?

- avoid compilation and dependencies problems: an environment manager will take care of everything!
- have several environments in parallel each with their own set of tools
- useful when cross-tools dependencies are incompatible with each other

CONDA : Access

Conda distribution

- Anaconda: a data science platform, comes with a lot of packages
- Miniconda: come without installed packages

Anconda cloud, the ”conda hub”

- Anaconda cloud (private company) relies on the community of developers, concerns many domains (Machine Learning, Data Visualization, Dashboarding-web, Image Processing, Natural Language Processing, etc)
- Anaconda cloud: made up of channels/owners. Each channels contains one or more conda packages
- be careful when downloading any packages from an untrusted source, always inspect before installation

About channels

Some conda channels

- default
- conda-forge: many popular python packages (analogous to PyPI but with a unified, automated build infrastructure and more peer review of recipes)
- bioconda: bioinformaticians' contributions
- private

Channels list order

- when different channels have the same package ⇒ collisions
- collisions resolved following the order of your channels list ⇒ put supplemental channels at the bottom of your channel list

R, mamba

Conda and R

The R interpreter is included in the r-essentials package (200 r packages).
Add r- before the regular r package name (eg. r-ggplot2)

Favorites	Downloads	Package (owner / package)	Platforms
18	304565	r / r-essentials 3.6.0 Some essential packages for working with R conda	linux-32 linux-64 osx-64 win-32 win-64
11	198513	skyblued / r-essentials 3.5.1 Some essential packages for working with R copy conda	linux-64 osx-64 win-32 win-64
1	66845	conda-forge / r-essentials 4.1 Some essential packages for working with R. This was migrated from the 'r' channel. conda	linux-64 noarch osx-64

Mamba

A fast drop-in alternative to conda, using libsolv for dependency resolution

```
conda install -c conda-forge mamba
```

Next, replace conda by mamba to use it

command

simple

```
conda create env -n myenv # creation of a conda environment info --envs #
conda list environments (* for the active one)activate myenv # active the
conda myenv environment
conda list # list packages (only in an active environment)install package
conda # installation of a tool/package remove package # suppress the
conda tool from the
    environment
conda env remove -n myenv # suppress the myenv environment
conda deactivate # inactivate the environment
```

miniconda3

With the miniconda3 distribution and by default, environments are installed in a miniconda3/envs/repository

2 modes

interactive

- create an environment
- activate the environment
- install some conda packages

configuration file

- list all conda packages in a configuration file (yamlor jsonformat)
- create the environment based on the configuration file (option -f)
- activate the environment

reproducibility

- good practice: use a configuration file
- specify a precise version of a package:
  <channel>::<package>=<version>

Conda Exercise

Conda setup

How to access conda?

- Conda is so used that it could even be installed by default to your machine. To test this: conda --version
- if not, may install it or got it by a docker image:

```
1 docker run -i -t -v ${PWD}:/data continuumio/miniconda3
```

- already activated on the IFB cluster (otherwise with module: module load conda)

How to access tools?

Manage Conda environment

1. create the working environment:

```
conda create env -n myenv
```

2. activate it:

```
conda activate myenv
```

3. if not yet done, install packages (specify the channel):

```
conda install -c bioconda bowtie2
```

4. work with the tools
5. quite the environment:

```
conda deactivate
```

Install snakemake with conda

Objective

Create a conda configuration file to install the snakemake tool.

Hint

- Search its channel in the Anaconda cloud web pages
- the ”minimal” environment is sufficient

Install snakemake with conda

condaEnvSmk.yml

```
channels:
   - conda-forge
   - bioconda
   - main
dependencies:
   - snakemake-minimal=6.5.0
```

run

```
conda env create -n condaEnvSmk -f condaEnvSmk.yml
conda activate condaEnvSmk
snakemake --version
```


docker

What is Docker?

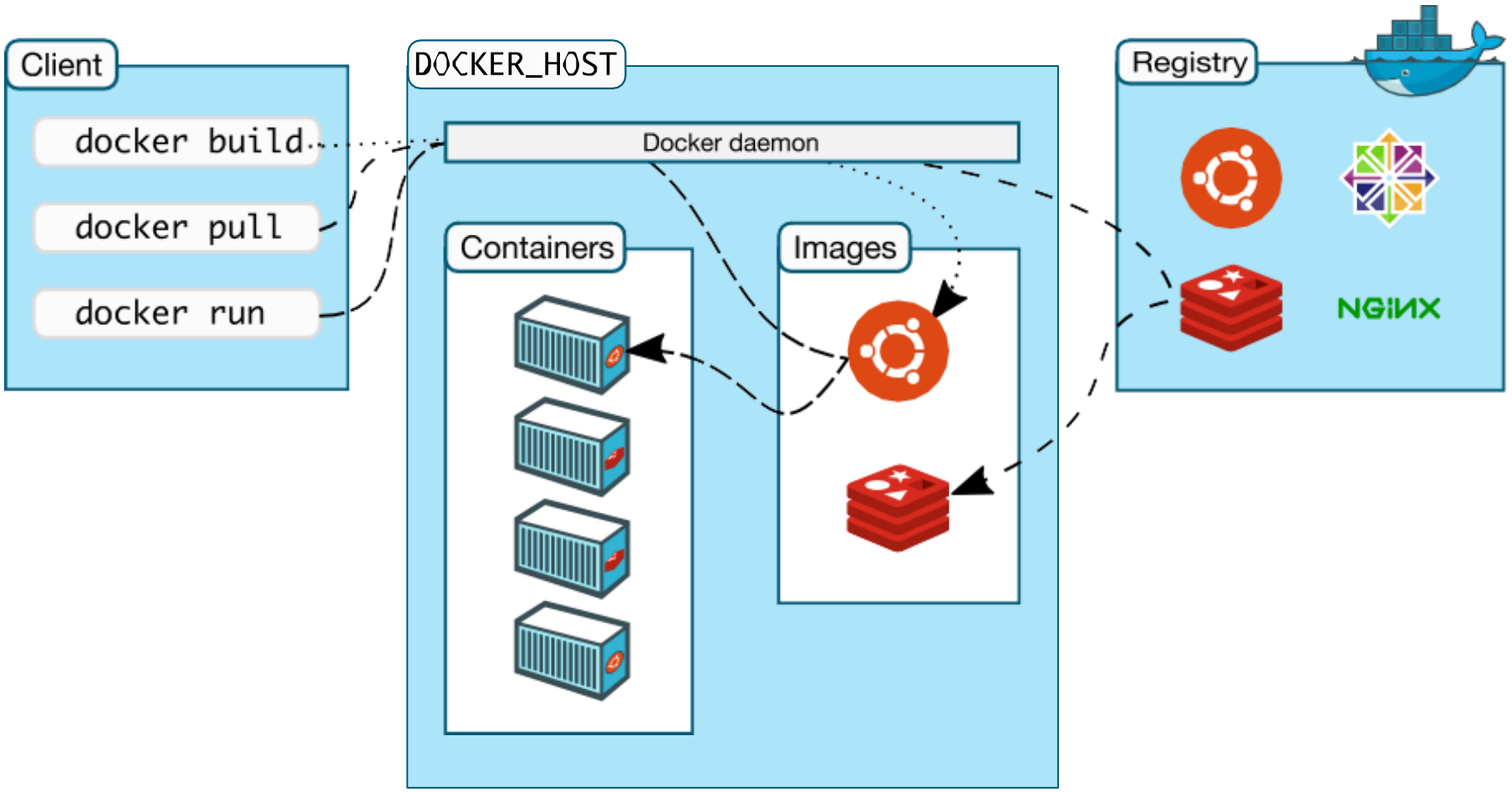
Docker is not very “old”

- First commit January 2013
- First version March 2013
- Version 1.0 in June 2014

But its adoption was fast

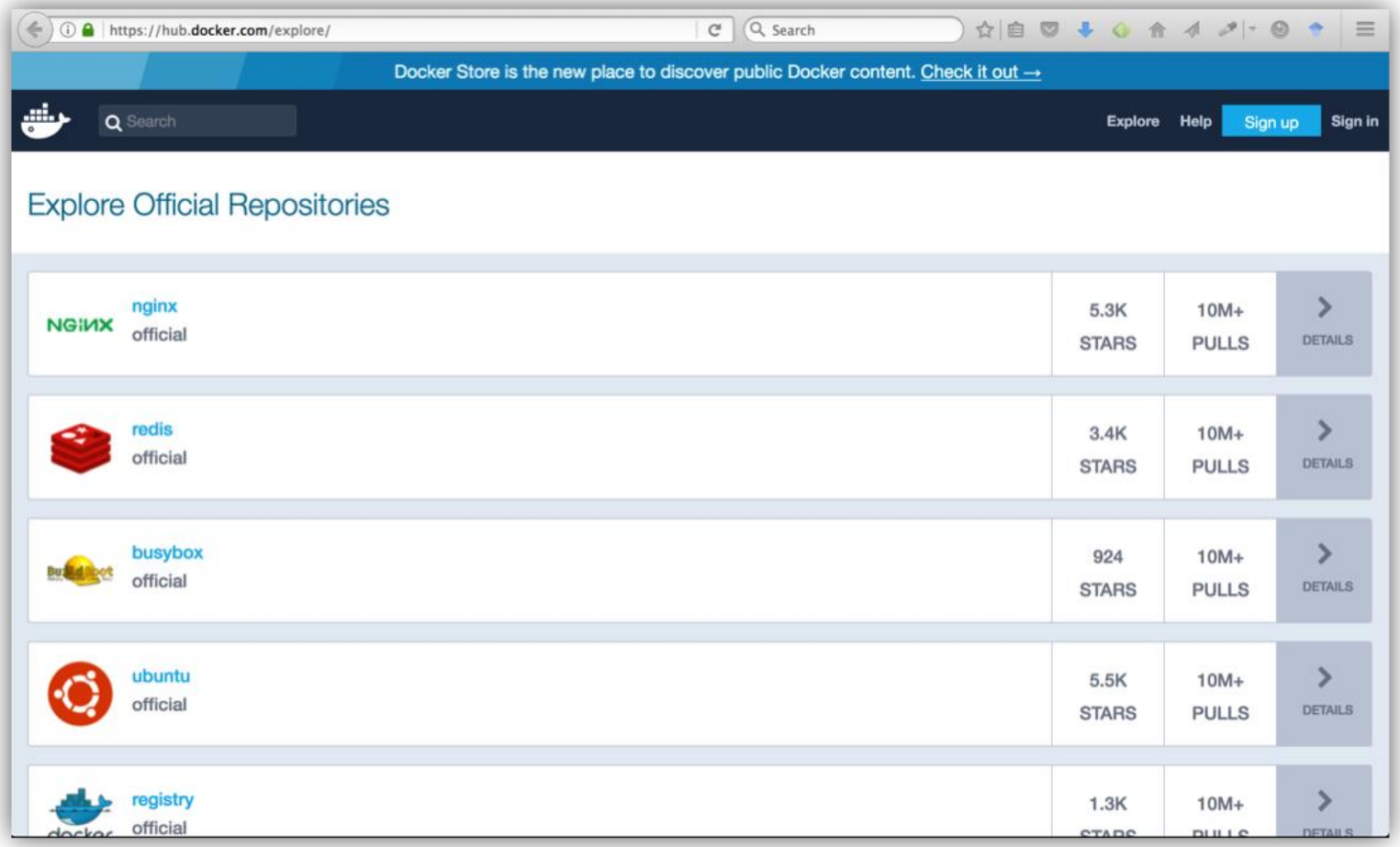
- Officially packaged in Ubuntu since 2014 (v14.04)

What is Docker?



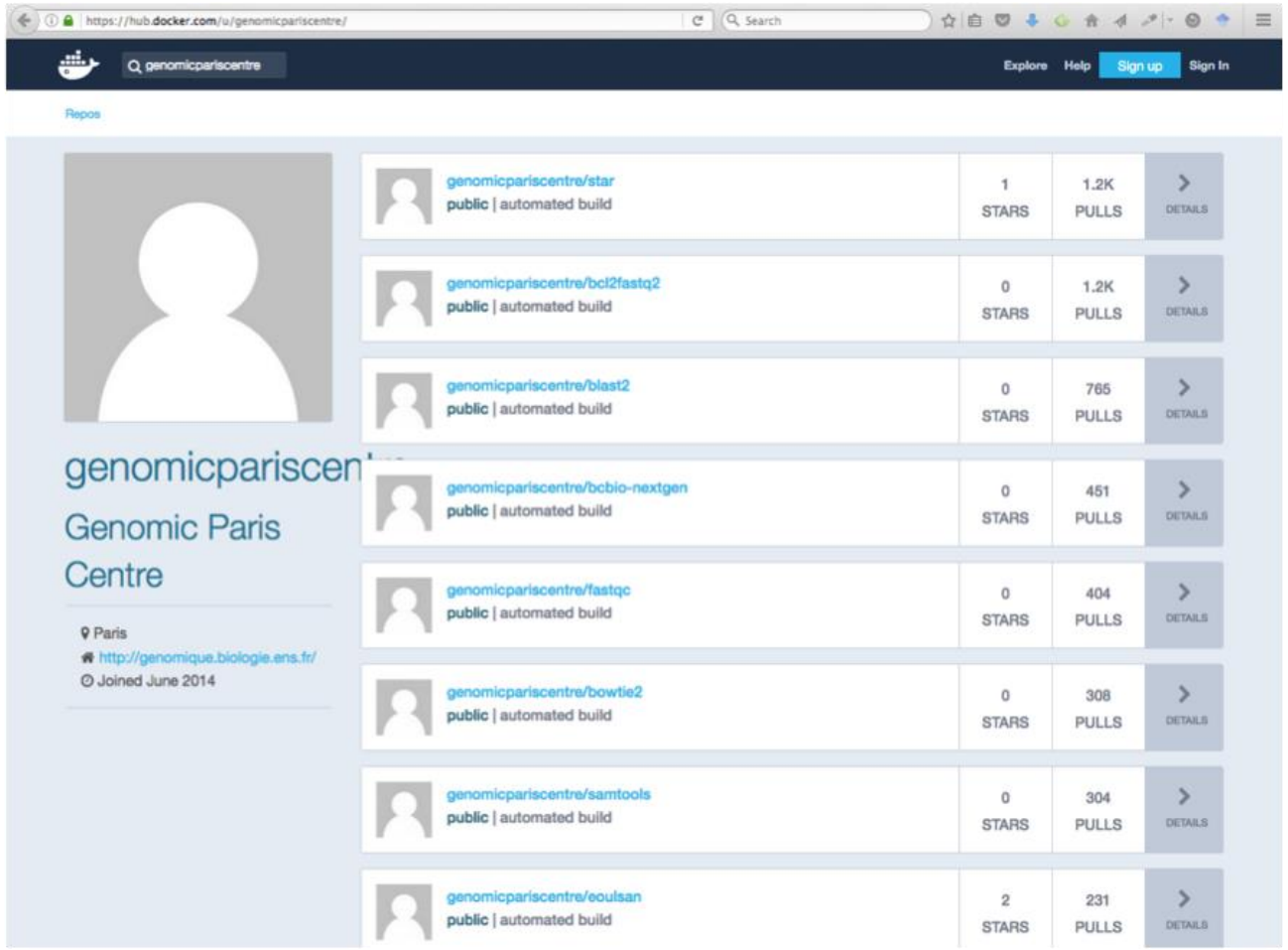
(https://docs.docker.com/get-started/overview/)

DockerHub



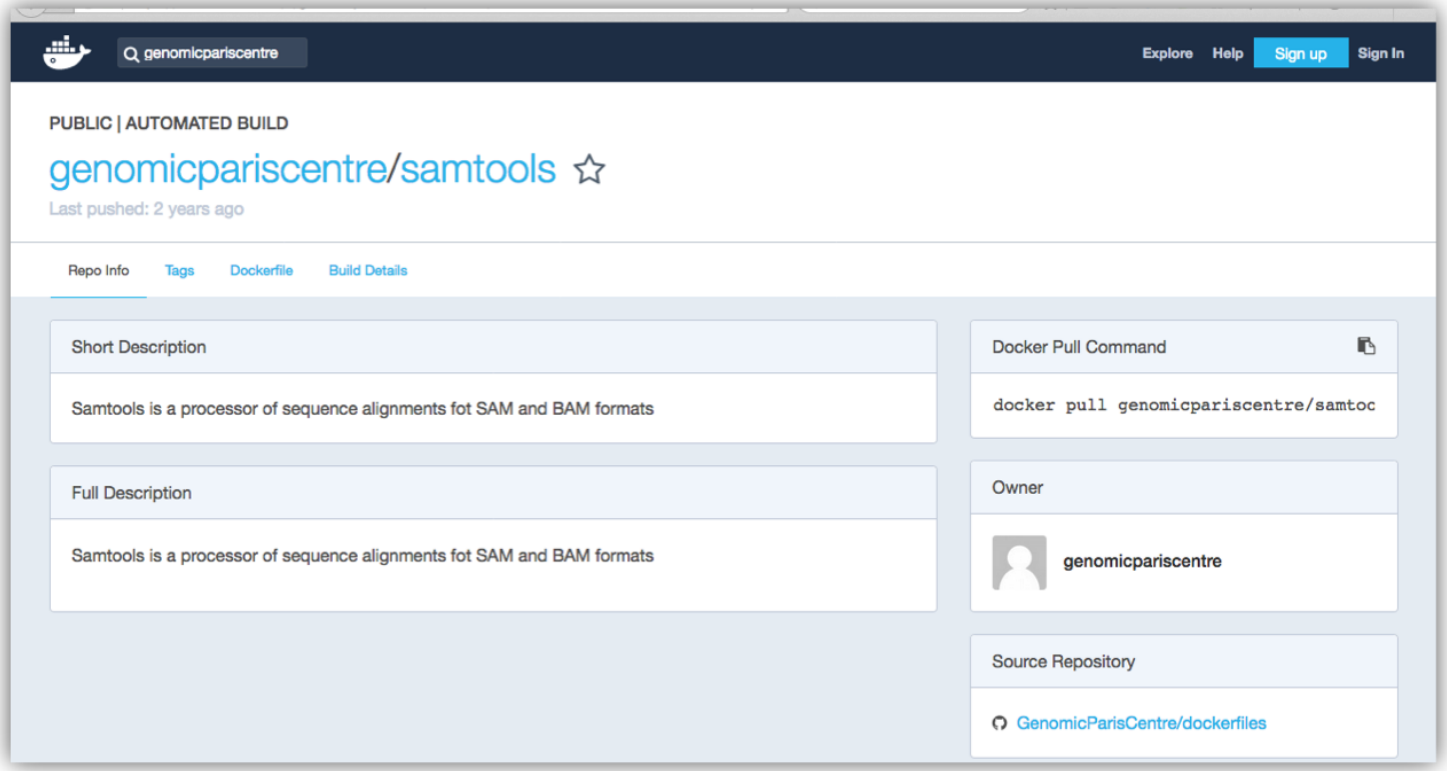
(https://hub.docker.com/)

DockerHub : Usermade images (1/2)

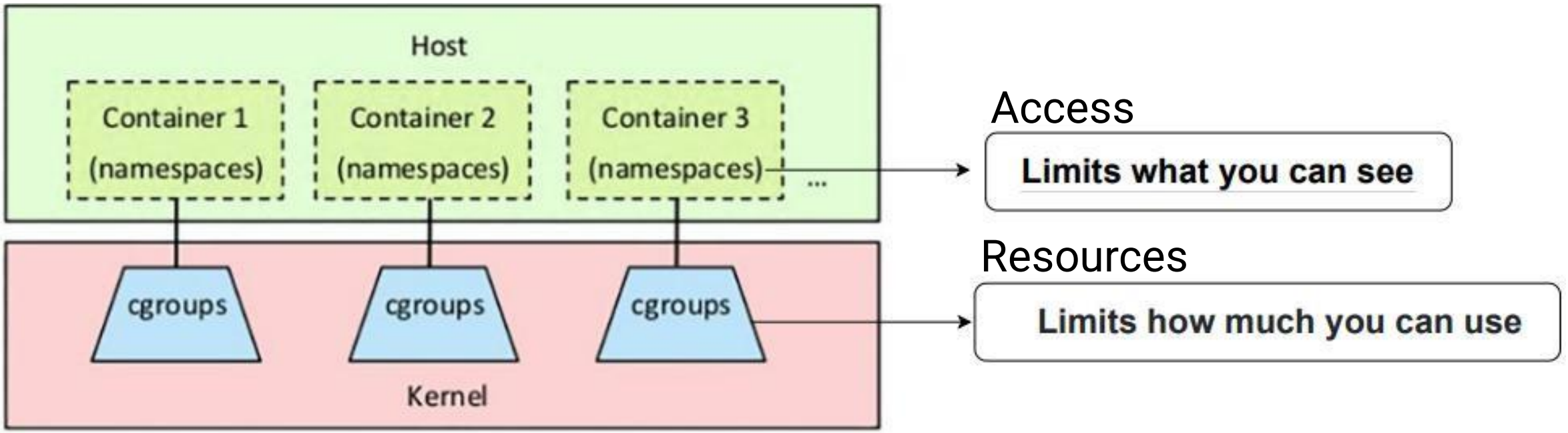


(urlhttps://hub.docker.com/u/genomicpariscentre/)

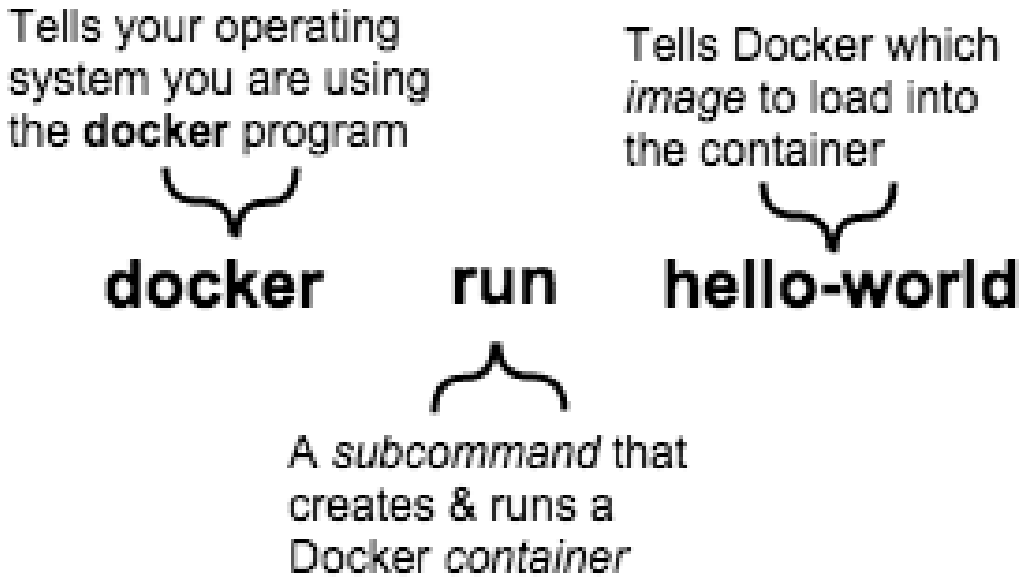
DockerHub : Usermade images (2/2)



(https://hub.docker.com/r/genomicpariscentre/samtools/)


Host
Container 1
(namespaces)
Container 2
(namespaces)
Container 3
(namespaces)
...
cgroups
cgroups
cgroups
Kernel
Access
Limits what you can see
Resources
Limits how much you can use

The client


Tells your operating system you are using the **docker** program
Tells Docker which *image* to load into the container
docker
run
hello-world
A *subcommand* that creates & runs a Docker *container*

Dockerfile

```
FROM ubuntu:20.04

LABEL maintainer = "Jason Charamis"
LABEL contact    = "jason.charamis@gmail.com"
LABEL build_date = "2023-11-30"
LABEL version    = "v.0.0.1-dev"


# Set environment variables for non-interactive installation
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe

# Step 1: Install essential dependencies
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        build-essential \
        libcurl4-openssl-dev \
        libssl-dev \
        libxml2-dev \
        git \
        python3-pip \
        gzip \
        wget

# Install Miniconda
RUN wget --quiet https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O /tmp/miniconda.sh && \
    /bin/bash /tmp/miniconda.sh -b -p /opt/conda && \
    rm /tmp/miniconda.sh

# Set conda on the PATH
ENV PATH /opt/conda/bin:$PATH
RUN conda init --all

# Step 2: Install R without prompts
RUN apt-get install -y --no-install-recommends \
        r-base \
        r-base-dev \
        && ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

"FROM": Base image

"LABEL": optional labels for Docker image

"RUN": Run commands in the terminal
(sudo rights are pre-defined)

Build your container as you would
for the OS of your laptop (e.g. libraries, Python)

"ENV" (paths) -> Set conda in $PATH

Typically used for installing dependencies
and setting tools in the PATH

How to see if a container is running ?

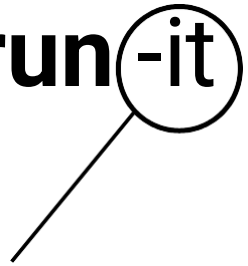
- sudo docker container ps -a (lists all containers)
- sudo docker container ls (lists active containers)

How to inspect my built Docker image?

sudo docker run -it --name rnaseq automated_variant_calling:latest

How to run a containerized application?

sudo docker **run** -it



run interactively

Docker commands

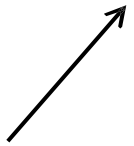
Other commands :

- docker images : list images available locally
- docker ps : status of containers
- docker rm : delete a container docker rmi : delete an image
- ...

(More details during the practical session.)

How to run a containerized application?

sudo docker **run** -it ***-v t(pwd):/mnt/workdir***



Bind my current directory with

container’s ‘/mnt/workdir’ directory

Ensure:

1. access to data
2. having the same paths

How to run a containerized application?

sudo docker **run** -it ***-v t(pwd):/mnt/workdir -w /mnt/workdir***

Bind my current directory with container's '/mnt/workdir' directory

Define the '/mnt/workdir' directory as the container's working directory

Ensure:

1. access to data
2. having the same paths

How to run a containerized application?

sudo docker run -it -v $(pwd):/mnt/workdir -w /mnt/workdir

automated_variant_calling:latest

How to run a containerized application?

sudo docker run -v $(pwd):/mnt/workdir -w /mnt/workdir automated_variant_calling:latest ***snakemake --cores 20 --snakefile AVC/workflow/Snakefile --use-conda --conda-frontend mamba --verbose --debug-dag***

Snakemake command

Hands-on project

Run example Variant calling analysis from E. coli data set using the containerized automated workflow

Let’s explain the order through the Snakefile...

Snakemake best-practices

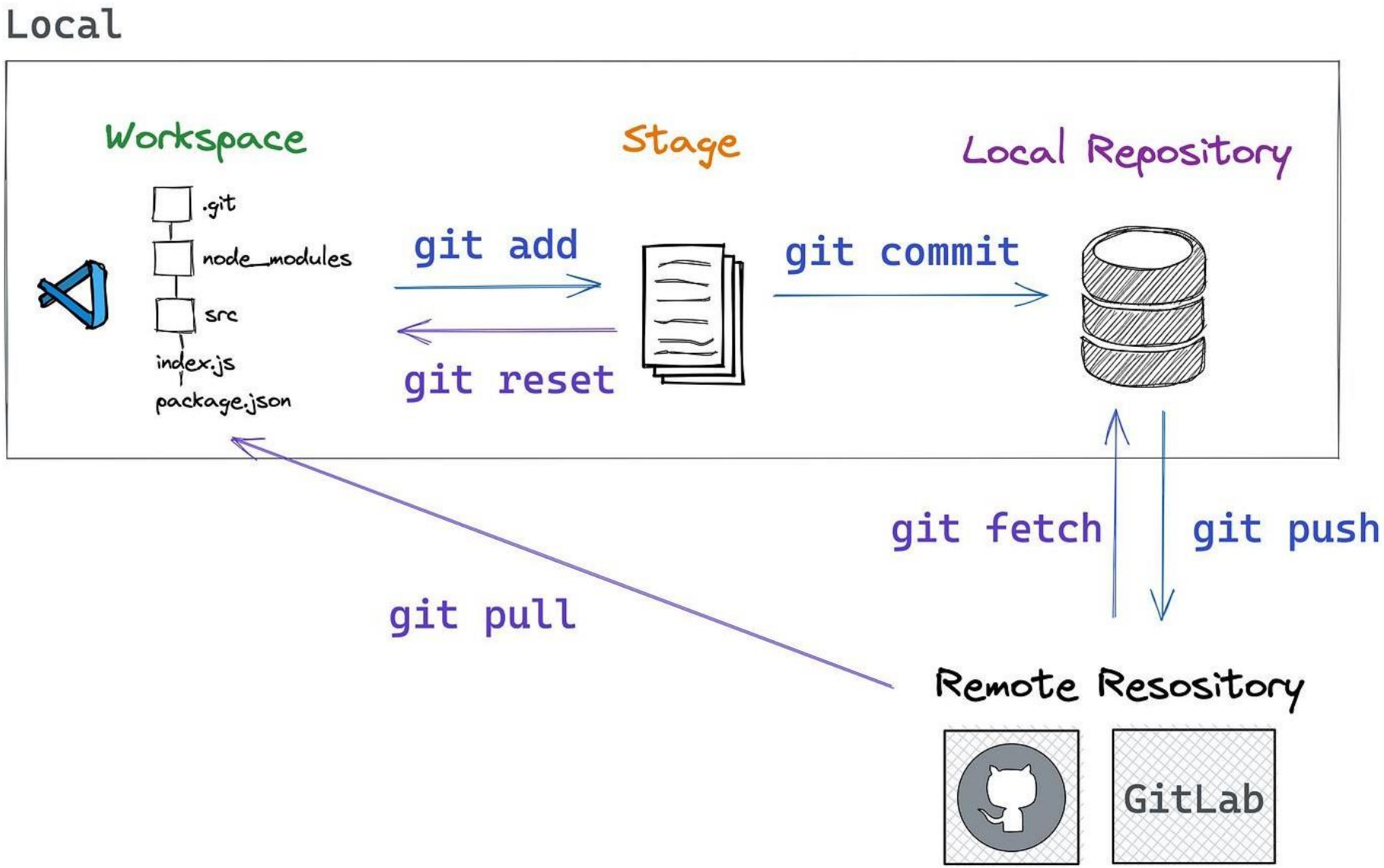
1. Snakemake wrappers for common NGS tools:

https://snakemake-wrappers.readthedocs.io/en/stable/wrappers/bwa-mem2/mem.html

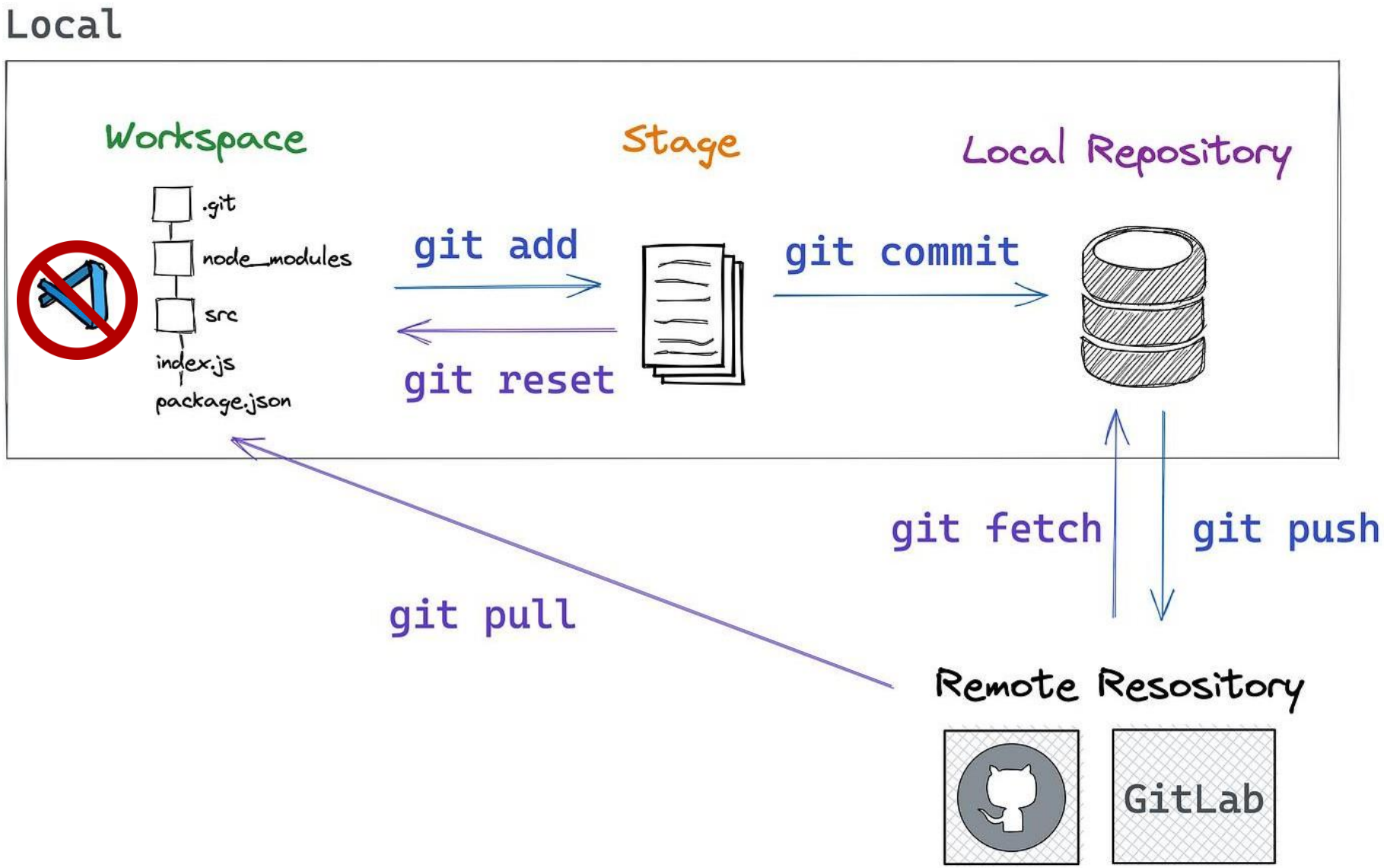
2. Snakemake executors (HPC, Cloud computing):

https://snakemake.readthedocs.io/en/v7.0.3/_modules/snakemake/executors.html

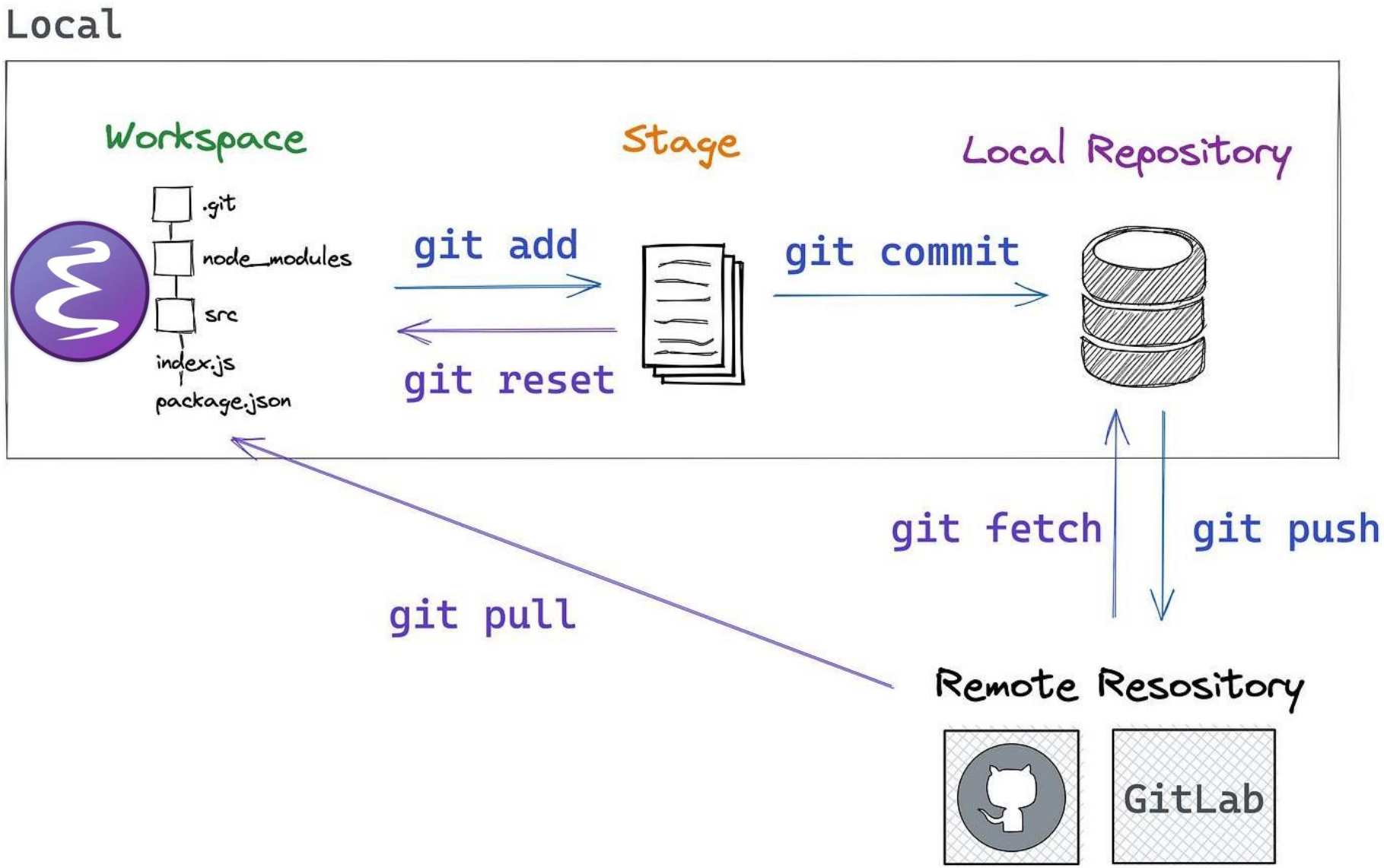
Git Version Control



Git Version Control



Git Version Control



git_commit.py

```
import os
import re
import subprocess
import argparse

def has_changes(repository):
    result = subprocess.run(["git", "status", "--porcelain"], cwd=repository, capture_output=True, text=True)
    return bool(result.stdout.strip())

def commit(repository, commit_message):
    if has_changes(repository):
        subprocess.run(["git", "add", "."], cwd=repository)
        subprocess.run(["git", "commit", "-m", commit_message], cwd=repository)
        subprocess.run(["git", "push", "origin", "main"], cwd=repository)
    else:
        print(f"No changes in the repository at {repository}. Skipping commit and push.")

def change_repositories_and_commit(start_repository, commit_message):
    repositories = set()

    for root, dirs, files in os.walk(start_repository):
        repositories.add(re.sub("\./|/.*","",root))
        repositories = { repo for repo in repositories if not re.search("\.",repo) }

    for d in repositories:
        commit ( d, commit_message )

def parse_arguments():
    parser = argparse.ArgumentParser(description='Commit and push changes to Git repositories.')
    parser.add_argument('-d', '--repository', required=False, type=str, help='Git repository you want to handle.')
    parser.add_argument('-cm', '--commit_message', required=False, type=str, help='Message for commit -m changes.')
    return parser.parse_args()

def main():
    args = parse_arguments()

    if args.repository:
        commit(args.repository, args.commit_message if args.commit_message else "Small corrections")
    else:
        start_repository = '.' # Start at parent directory of repositories
        change_repositories_and_commit(start_repository, args.commit_message if args.commit_message else "Small corrections")

if __name__ == "__main__":
    main()
```

DevOps philosophy of software development

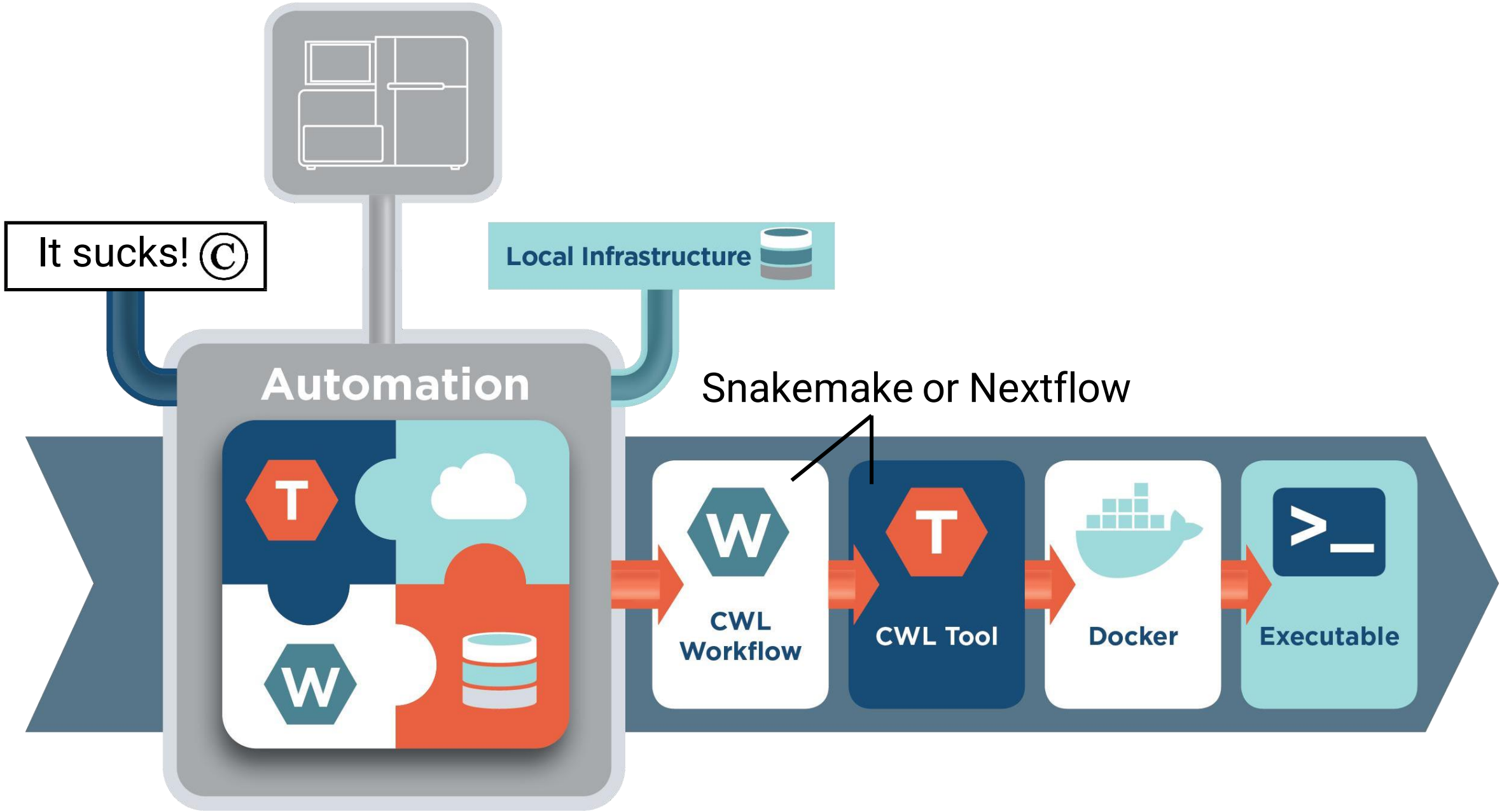
Code changes continously integrated into the testing, pre-production and production stage

- Agile Development Practices
- Continuous Integration / Continuous Deployment

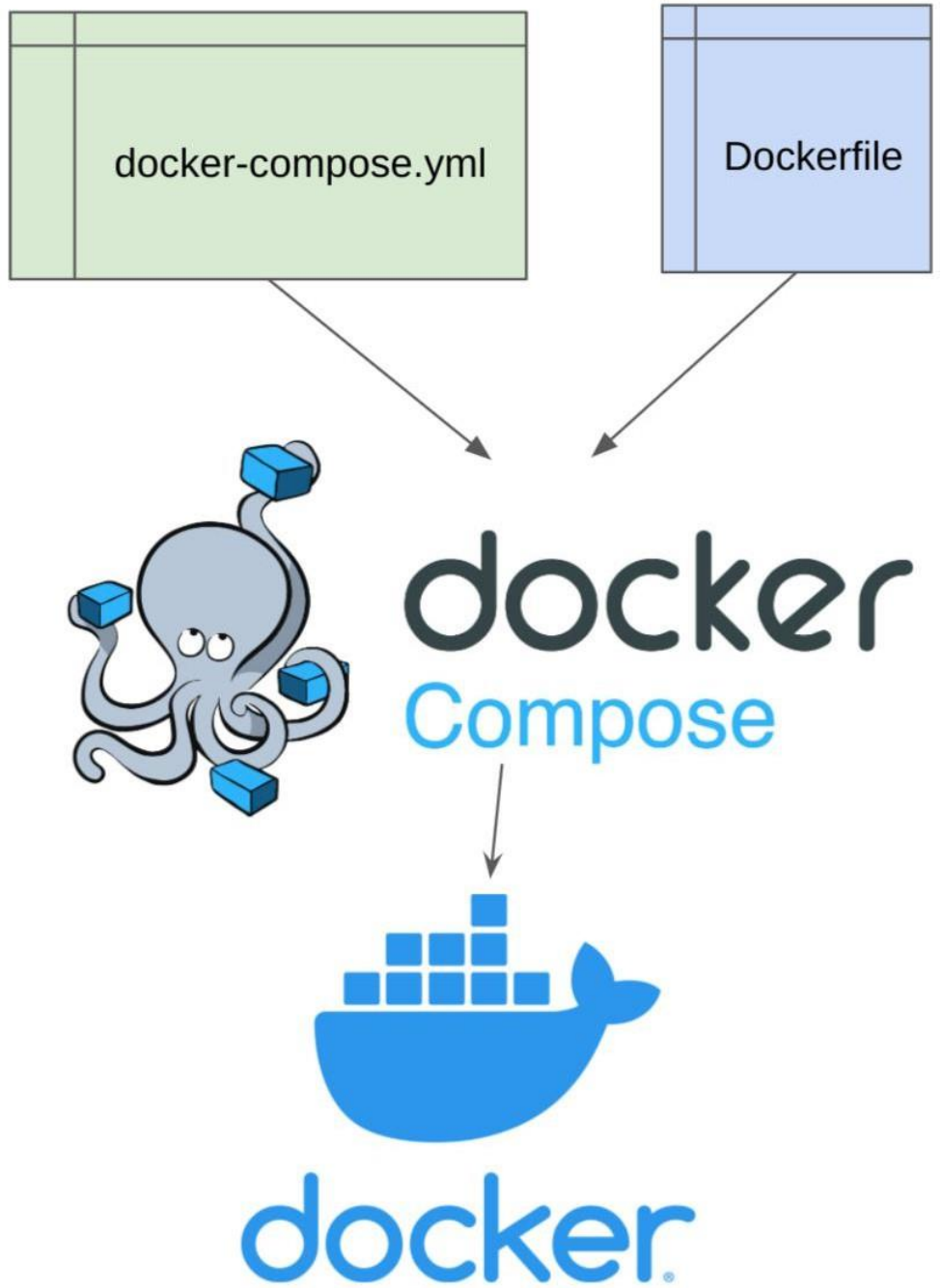
BASH

THE BOURNE-AGAIN SHELL

The Missing Semester of your CS education:
https://missing.csail.mit.edu/2020/

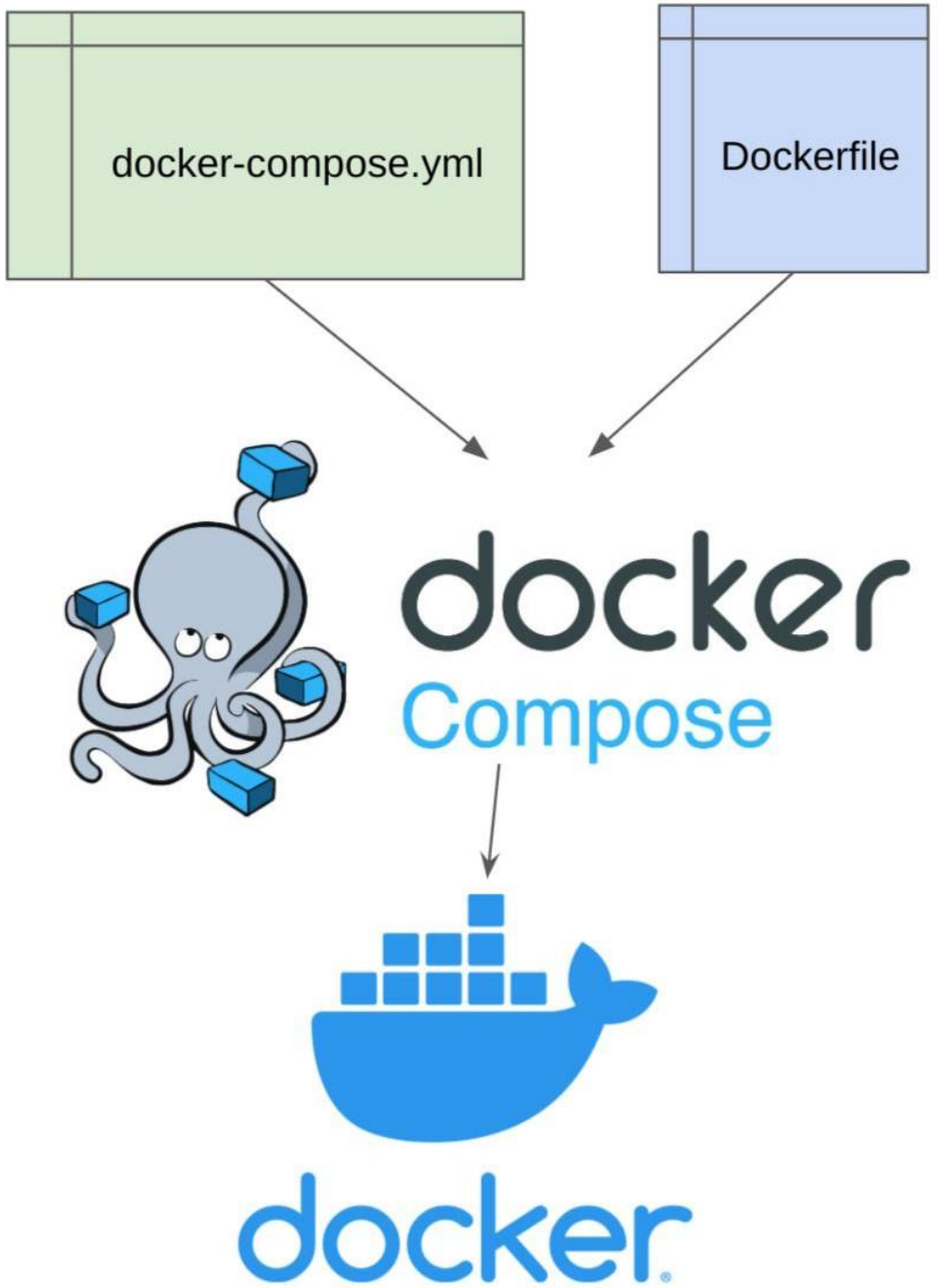

It sucks! ©
Local Infrastructure
Automation
Snakemake or Nextflow
CWL Workflow
CWL Tool
Docker
Executable

- Combine multiple Docker containers to create modern data services



- Combine multiple Docker containers to create modern data services

- Kubernetes used for managing container-based systems



Suggestion - 1

1. Build a Snakemake/Nextflow pipeline

2. Containerize it (Docker)

3. Upload to GitHub or Gitlab

Suggestion - 2

Find a pipeline/project **<u>you actually care about</u>**

NGS

1. RNAseq, scRNAseq
2. DNAseq (Structural Variants)
    - Short reads, Long reads
3. ATACseq
4. ChipSeq
5. MethylSeq (AVCmark)
6. HiC

Computational Phylogenetics

1. Maximum Likelihood phylogenetic analysis
2. Identification of positive selection

Great resources

- Snakemake workshop by Romain Feron:

https://github.com/RomainFeron/workshop-snakemake-sibdays2022

Ευχαριστώ