

A peer-reviewed version of this preprint was published in PeerJ on 24 September 2015.

[View the peer-reviewed version](https://doi.org/10.7717/peerj.1273) (peerj.com/articles/1273), which is the preferred citable publication unless you specifically need to cite this preprint.

Di Tommaso P, Palumbo E, Chatzou M, Prieto P, Heuer ML, Notredame C. 2015. The impact of Docker containers on the performance of genomic pipelines. PeerJ 3:e1273 <https://doi.org/10.7717/peerj.1273>

The impact of Docker containers on the performance of genomic pipelines

Paolo Di Tommaso^{*1,2}, Emilio Palumbo^{1,2}, Maria Chatzou^{1,2}, Pablo Prieto^{1,2}, Michael L Heuer³,
Cedric Notredame^{1,2}

¹Comparative Bioinformatics, Bioinformatics and Genomics Program,
Centre for Genomic Regulation (CRG), Dr. Aiguader 88, 08003 Barcelona, Spain

²Universitat Pompeu Fabra (UPF), 08003 Barcelona, Spain

³National Marrow Donor Program, Minneapolis, MN 55413-1753, USA

* Corresponding author: paolo.ditomaso@crg.eu

Abstract

Genomic pipelines consist of several pieces of third party software and, because their experimental nature, frequent changes and updates are commonly necessary thus raising serious distribution and reproducibility issues. Docker containers technology offers an ideal solution, as it allows the packaging of pipelines in an isolated and self-contained manner. This makes it easy to distribute and execute pipelines in a portable manner across a wide range of computing platforms. Thus the question that arises is to what extent the use of Docker containers might affect the performance of these pipelines. Here we address this question and conclude that Docker containers have only a minor impact on the performance of common genomic pipelines, which is negligible when the executed jobs are long in terms of computational time.

Introduction

Genomic pipelines usually rely on a combination of several pieces of third party research software. These applications tend to be academic prototypes that are often difficult to install, configure and deploy. Furthermore their experimental nature can result in frequent updates, thus raising serious reproducibility issues. In the past virtual machines were proposed as an answer to this issue. They are indeed very convenient but come along with a few major issues that include high latency and significant overhead.

Docker containers technology has been designed to address these issues. It has recently received an increasing level of attention throughout the scientific community because it allows applications to run in an isolated, self-contained package that can be efficiently distributed and executed in a portable manner across a wide range of computing platforms.

The first most obvious advantage of this approach is to replace the tedious installation of numerous pieces of software, with complex dependencies, by simply downloading a single pre-built ready-to-run image containing all the software and the required configuration.

The second strength of Docker is to run each process in an isolated container that is created starting from an immutable image. This prevents conflicts with any other installed program in the hosting computing environment, and guarantees that each process runs in a predictable system configuration that cannot change over time due to misconfigured software, system updates or programming errors.

Containers only require a few milliseconds to start and many instances can run in the same hosting environment. This is possible because it runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.

A study from IBM Research showed that Docker technology introduces a negligible overhead for CPU and memory performance, and applications running in a container perform equally or better when compared to KVM virtualization in all tests (1).

43 In this work we assess the impact of Docker containers on the performance of genomic pipelines
44 using a realistic computational biology usage scenario based on the re-computation of selected
45 subsets of the ENCODE analysis.

46 **Method**

47 In order to evaluate the impact of Docker usage on the execution performance of bioinformatics tools
48 we benchmarked three different genomic pipelines. A comparison of the execution times was made
49 running them with and without Docker along with the same dataset. The tests were executed using a
50 cluster node HP BL460c Gen8 with 12 cpus Intel Xeon X5670 (2.93GHz), 96 GB of RAM and running
51 on Scientific Linux 6.5 (kernel 2.6.32-431.29.2.el6.x86_64).

52 Tests were executed using Docker 1.0 configured with "device mapper" as the storage driver. Docker
53 images used for the benchmark were built starting from a Scientific Linux 6.5 base image (2). The
54 compute node was reserved for the benchmark execution (this means that no other workload was
55 dispatched to it), moreover to prevent any possible network latencies that could affect the execution
56 times in a aleatory manner, all tests were executed using the node local disk as main storage.

57 All three pipelines are developed with Nextflow, a tool that is designed to simplify the deployment of
58 computational pipelines across different platforms in a reproducible manner (4). Nextflow integrates
59 the support for Docker allowing pipeline tasks to be executed transparently in Docker containers.

60 This allowed us to execute the same pipeline natively or run it with Docker without having to modify
61 the pipeline code, but by simply specifying the Docker image to be used in the Nextflow configuration
62 file.

63 It should be noted that when the pipeline is executed with Docker support it does not mean that the
64 overall pipeline execution runs "inside" a single container, but that each task spawned by the pipeline
65 runs in its own container. This approach allows a Docker based pipeline to use a different image for
66 each different task in the computational workflow, and therefore scale seamlessly in a cluster of
67 computers (which wouldn't be possible using the single container approach).

68 The overhead introduced by containers technology on the pipelines performance was estimated by
69 comparing the median execution time of 10 instances running with and without Docker. As the
70 pipeline ran parallel tasks, the execution time was normalized summing up the execution time of all
71 the tasks in each instance.

72 **Benchmark 1**

73 The first performance evaluation was carried out using a simple pipeline for RNA-Seq data analysis
74 (15).

75 The pipeline takes raw RNA-Seq sequences as input and first maps them to a reference genome and
76 a transcript annotation by sequence alignment. The mapping information is then used to quantify
77 known transcripts using the reference transcript annotation. For each processed sample, the pipeline
78 produces as output a table of relative abundances of all transcripts in the transcript annotation.

79 The pipeline was run 10 times using the same dataset with and without Docker. The RNA-Seq data
80 was taken from the ENCODE project and contained randomly sampled (10% of the original) Illumina
81 paired-end sequences from brain samples (CNS) of mouse embryos at day 14 and day 18, in 2
82 bioreplicates. Each run executed a first *index* task using Bowtie, then a *mapping* task using Tophat2
83 and finally a *transcript* task using the Cufflinks tool. The following versions of these tools were used:
84 Samtools 0.1.18 (5), Bowtie2 2.2.3 (6), Tophat-2.0.12 (7), Cufflinks 2.2.1 (8).

85 Each run executed 9 tasks. The median pipeline execution time in the native environment was
86 1,158.4 minutes (19h 18m 23s), while the median execution time when running it with Docker was

87 1,157.6 minutes (19h 17m 35s). Thus, the use of Docker containers didn't add any time overhead to
 88 the pipeline execution, on the contrary the median execution time was a few seconds faster (0.1%).

Pipeline	Tasks	Mean task time (mins)		Median execution time (mins)		Slow down
		native	docker	native	docker	
RNA-Seq	9	128.7	128.6	1,158.4	1,157.6	0.999
Variant call.	48	26.1	26.7	1,252.6	1,283.6	1.025
Piper	98	0.6	1.0	58.5	97.1	1.659

89 **Table 1**

90 **Benchmark 2**

91 The second benchmark was executed using an assembly-based variant calling pipeline (16), part of a
 92 Minimum Information for Reporting Next Generation Sequencing Genotyping (MIRING)-compliant
 93 genotyping workflow for histocompatibility, immunogenetic and immunogenomic applications (3).

94 Paired-end genomic reads from targeted human leukocyte antigen (HLA) and killer-cell
 95 immunoglobulin-like receptors (KIR) genes are assembled into consensus sequences. Reads and
 96 consensus sequences are then aligned to the human genome reference and used to call variants.

97 The pipeline was launched 10 times using Illumina paired end genomic reads targeted for major
 98 histocompatibility complex (MHC) class I HLA-A, HLA-B, and HLA-C genes and MHC class II gene
 99 HLA-DRB1 from 8 individuals. The following versions of these tools were used both in the native and
 100 Docker environment: ngs-tools 1.7 (14), SSAKE 3.8.2 (9), BWA 0.7.12-r1039 (10), Samtools 1.2 (5).

101 Each run executed 48 tasks, and the maximum number of tasks that could be executed in parallel
 102 was set to 10. Most of the tasks completed in a few seconds, with the exclusion of the SSAKE stage
 103 which needed from 2 to 3.5 hours to complete (see fig. 2).

104 The median pipeline execution time in the native environment was 1,252.6 minutes (20h 52m 34s),
 105 while the median execution time when running it with Docker was 1,283.6 minutes (21h 23m 38s).
 106 This means that when running with Docker the execution was slowed down by 2.5% (see table 1).

107 **Benchmark 3**

108 The last benchmark was carried out using Piper-NF, a genomic pipeline for the detection and
 109 mapping of long non-coding RNAs (17).

110 The pipeline takes as input cDNA transcripts sequences in FASTA format which are blasted against a
 111 set of genomes also provided in FASTA format. Homologous regions on the target genomes are used
 112 as anchor points and the surrounding regions are then extracted and re-aligned with the original
 113 query. If the aligner can align these sequences and the alignment covers a required minimal region of
 114 the original query, the sequences are used to build a multiple sequence alignment which is then used
 115 to obtain the similarity between each homologous sequence and the original query.

116 As in previous experiments the pipeline was run 10 times using the same dataset with and without
 117 Docker. We used as the input query a set of 100 RNA-Seq transcript sequences in FASTA format
 118 from Gallus gallus species. The input sequences were mapped and aligned towards a set of genomes
 119 consisting of Anas platyrhynchos, Anolis carolinensis, Chrysemys picta bellii, Ficedula albicollis,
 120 Gallus gallus, Meleagris gallopavo, Melospiza undulatus, Pelodiscus sinensis, Taeniopygia
 121 guttata, from Ensembl version 73. The following versions of the tools were used both in the native and

122 Docker environment: T-Coffee 10.00.r1613 (11), NCBI BLAST 2.2.29+ (12), Exonerate 2.2.0 (13).
123 Each run executed 98 jobs and the maximum number of tasks that could be executed in parallel was
124 set to 10.

125 The median pipeline execution time in the native
126 environment was 58.5 minutes, while the median
127 execution time when running it with Docker was 97.1
128 minutes. In this experiment running with Docker
129 introduced a significant slowdown of the pipeline
130 execution time, around 66% (see table 1).

131 This result can be explained by the fact that the
132 pipeline executed many short-lived tasks: the mean
133 task execution time was 35.8 seconds, and the median
134 execution time was 5.5 seconds (see fig. 3). Thus the
135 overhead added by Docker to bootstrap the container
136 environment and mount the host file system became
137 significant when compared to the short task duration.

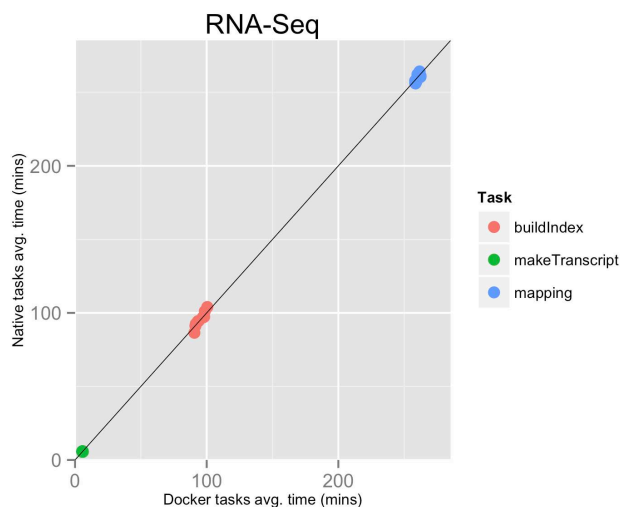


Figure 1

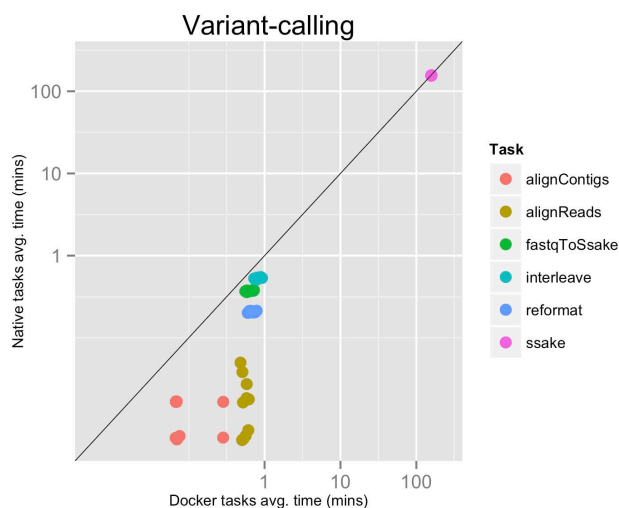


Figure 2

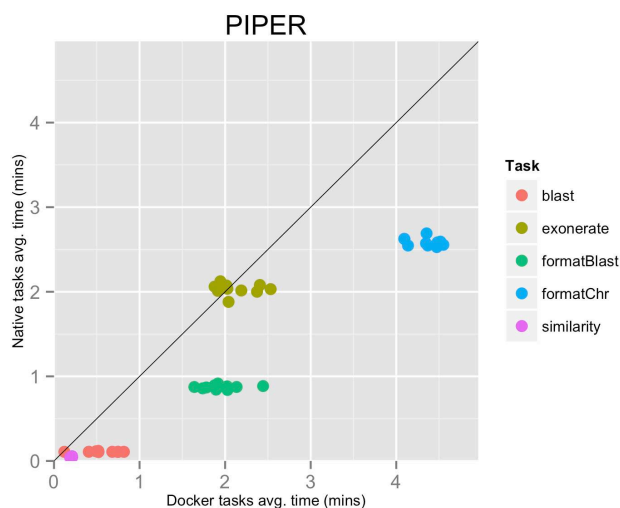


Figure 3

138 Results

139 In this work we assessed the impact of Docker containers technology on the performance of genomic
140 pipelines. We showed that container "virtualization" has a negligible overhead on pipeline
141 performance when it is composed by medium/long running tasks, which is the most common scenario
142 in computational genomic pipelines. While the performance degradation is more significant for
143 pipelines where most of the tasks have a fine or very fine granularity (few seconds or milliseconds).

144 Conclusion

145 The fast start-up time for Docker containers technology allows one to virtualize a single process or the
146 execution of a bunch of applications, instead of a complete operating system. This opens up new
147 possibilities, for example the possibility to "virtualize" distributed job executions in an HPC cluster of
148 computers.

149 In this work we show that Docker containerization has a negligible impact on the execution
150 performance of common genomic pipelines where tasks are generally very time consuming.

151 The minimal performance loss introduced by the Docker engine is offset by the advantages of running
152 an analysis in a self-contained and precisely controlled runtime environment. Docker makes it easy to
153 precisely prototype an environment, maintain all its variations over time and rapidly reproduce any
154 former configuration one may need to re-use. These capacities guarantee consistent results over time
155 and across different computing platforms.

156 References

- 157 1. Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. An Updated Performance
158 Comparison of Virtual Machines and Linux Contain. IBM Research (2014). Available at
159 <http://ibm.co/V55Otg> (accessed 1 Jun 2015)
- 160 2. Scientific Linux 6.5 Docker image. Available at <https://registry.hub.docker.com/u/ringo/scientific/>
161 (accessed 5 May 2015)
- 162 3. Steven J. Mack, et al. Minimum Information for Reporting Next Generation Sequence Genotyping
163 (MIRING): Guidelines for Reporting HLA and KIR Genotyping via Next Generation Sequencing.
164 Available at <http://biorxiv.org/content/early/2015/02/16/015230> (accessed 1 Jun 2015)
- 165 4. Di Tommaso P., et al. Nextflow: A novel tool for highly scalable computational pipelines. (2014)
166 Available at <http://dx.doi.org/10.6084/m9.figshare.1254958>
- 167 5. Heng Li, et al. 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 25 (16):
168 2078-2079. doi: 10.1093/bioinformatics/btp352
- 169 6. Ben Langmead, Steven L Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nat*
170 *Methods*. 2012 Mar 4; 9(4): 357–359. doi: 10.1038/nmeth.1923
- 171 7. Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, Salzberg SL. TopHat2: accurate alignment of
172 transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biol*. 2013 Apr
173 25;14(4):R36. doi: 10.1186/gb-2013-14-4-r36.
- 174 8. Cole Trapnell, et al. Transcript assembly and quantification by RNA-Seq reveals unannotated
175 transcripts and isoform switching during cell differentiation. *Nature Biotechnology* 28, 511–515 (2010)
176 doi:10.1038/nbt.1621
- 177 9. Warren RL, Sutton GG, Jones SJM, Holt RA. Assembling millions of short DNA sequences using
178 SSAKE. *Bioinformatics* (2007) 23 (4): 500-501. doi: 10.1093/bioinformatics/btl629
- 179 10. Heng Li, Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler
180 transform. *Bioinformatics* (2009) 25 (14): 1754-1760. doi: 10.1093/bioinformatics/btp324
- 181 11. Notredame C, Higgins DG, Heringa J. T-Coffee: A novel method for fast and accurate multiple
182 sequence alignment. *J Mol Biol*. (2000) Sep 8;302(1):205-17. doi:10.1006/jmbi.2000.4042
- 183 12. Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. Basic local alignment search tool.
184 *J Mol Biol*. (1990) Oct 5; 215:403-410. PMID: 2231712
- 185 13. Guy St C Slater, Ewan Birney. Automated generation of heuristics for biological sequence
186 comparison. *BMC Bioinformatics* (2005) 6:31 doi:10.1186/1471-2105-6-31
- 187 14. NGS Tools. Available at <https://github.com/nmdp-bioinformatics/ngs>
- 188 15. RNA-Seq toy. Available at <https://github.com/nextflow-io/rnatoy/tree/docker-benchmark>
- 189 16. Consensus assembly and variant calling workflow.
190 Available at <https://github.com/nextflow-io/nmdp-flow/tree/docker-benchmark>
- 191 17. Piper-NF. Available at <https://github.com/cbcrg/piper-nf/tree/docker-benchmark>