CMK: Enhancing Resource Usage Monitoring across Diverse Bioinformatics Workflow Management Systems

Robert Nica[1,2*], Stefan Götz[2] and Germán Moltó[1*]

[1]Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC - Universitat Politècnica de València, Camino de Vera s/n, Valencia, 46022, Spain.
[2]BioBam Bioinformatics S.L., Avenida Peris y Valero 78-23, Valencia, 46006, Spain.

*Corresponding author(s). E-mail(s): bogni@doctor.upv.es; gmolto@dsic.upv.es;
Contributing authors: sgoetz@biobam.com;

Abstract

The increasing use of multiple Workflow Management Systems (WMS) employing various workflow languages and shared workflow repositories enhances the open-source bioinformatics ecosystem. Efficient resource utilization in these systems is crucial for keeping costs low and improving processing times, especially for large-scale bioinformatics workflows running in cloud environments. Recognizing this, our study introduces a novel reference architecture, Cloud Monitoring Kit (CMK), for a multi-platform monitoring system. Our solution is designed to generate uniform, aggregated metrics from containerized workflow tasks scheduled by different WMS. Central to the proposed solution is the use of task labeling methods, which enable convenient grouping and aggregating of metrics independent of the WMS employed. This approach builds upon existing technology, providing additional benefits of modularity and capacity to seamlessly integrate with other data processing or collection systems. We have developed and released an open-source implementation of our system, which we evaluated on Amazon Web Services (AWS) using a transcriptomics data analysis workflow executed on two scientific WMS. The findings of this study indicate that CMK provides valuable insights into resource utilization. In doing so, it paves the way for more efficient management of resources in containerized scientific workflows running in public

cloud environments, and it provides a foundation for optimizing task configurations, reducing costs, and enhancing scheduling decisions. Overall, our solution addresses the immediate needs of bioinformatics workflows and offers a scalable and adaptable framework for future advancements in cloud-based scientific computing.

Keywords: Cloud computing, Monitoring, High-throughput computing, Workflow Management Systems, Bioinformatics Infrastructure

1 Introduction

Generating increasingly large and complex datasets has become standard in modern biological research. Cloud computing, available from providers like Amazon Web Services (AWS) [1], Google Cloud [2], or Microsoft Azure [3], empowers research communities to self-provision customized virtual computing infrastructures. These public Cloud providers allocate computing, storage, networking, and services to users, allowing them to perform computations on a pay-as-you-go basis. The capacity, versatility, and ability to provision on-demand computing resources make cloud computation services appealing to researchers handling big data, such as bioinformaticians.

Complementary to cloud computing, the transition from Virtual Machines (VM) to software containers has marked a significant shift in scientific computing. VMs are commonly used in Cloud platforms to provide the computation substrate on which applications are executed, together with the isolation boundary, according to a set of predefined templates that specify the capabilities regarding numbers of virtual CPUs and disk storage, among other criteria. However, in recent years, a lighter kind of virtualization, called software containers [4], has become the de facto standard for running scientific algorithms [5]. Benefits of running software bundled in containers include minimal overhead [6], compared to virtual machines, and inclusion of all dependencies necessary for running the software; these aspects streamline the execution of tools and scripts created in different programming languages like R, Python, or Perl [7, 8].

In the context of bioinformatics, Cloud computing and container technology are often used synergistically to improve workflows or pipelines, offering scalable resources and reproducible environments for the efficient handling of complex sequences of data analysis tasks. Reproducibility, an aspect of research that becomes increasingly difficult with more complex data analysis tasks, is essential for validating scientific findings and ensuring the integrity of the study. Bioinformaticians, in particular, often face challenges associated with sharing fully reproducible analysis pipelines [9]. Established best practices for ensuring data reproducibility include the use of standardized workflow languages, such as the Common Workflow Language (CWL) [10], or the Workflow Description Language (WDL) [11], all of which provide the ability to define complex computational tasks in a structured, interoperable format and share workflows across different computing environments without the need to alter the underlying code. Moreover, adherence to the FAIR Data Principles and FAIR Computational Workflows [12] allows researchers to identify processing-specific requirements. An additional measure

to help ensure data reproducibility, provenance tracking [13], captures the entire life cycle of the data in a workflow [14, 15]; however, it has yet to be adopted broadly. Finally, most of the software tools used in science are available in software containers that bundle dependencies and can be executed with simple commands or combined in workflows with other tools. These scientific workflows are often shared with the scientific community through workflow repositories such as Dockstore [16] or WorkflowHub [17].

It is within this context that Workflow Management Systems (WMS) like Toil [18], Cromwell [11], miniwdl [19], or Nextflow [20], become integral. Such WMS offer a way to accurately automate and orchestrate the execution of complex workflows and process large amounts of data. Furthermore, they are compatible with standard systems of clusters and cloud computing services and can therefore be leveraged in analyses of large datasets. The most commonly adopted cloud services for WMS task computation are batch systems like AWS Batch [21], Azure Batch [22], and Google Batch [23], which dynamically allocate resources to meet the requirements of the analysis tasks. WMS acts as an easily accessible entry point for the user and hides the complexity of the analysis on the back-end. A WMS can run as a server (e.g., Cromwell and Galaxy) [24], where one instance of the WMS controls the execution of many workflows, or as a head task (e.g., Nextflow and miniwdl), where many WMS instances run in parallel with each controlling the execution of a single workflow.

To enhance the functionality and interoperability of WMS like those previously described, efforts have been made to establish uniform communication protocols within the bioinformatics computational environment (Fig. 1). The Global Alliance for Genomics & Health (GA4GH) has played a key role in this effort through the introduction of their Task Execution Schema (TES) [25], which specifies a communication protocol between the WMS and the computation environment. While direct implementation of TES by Cloud services is still pending, open-source TES server implementations, like Funnel [26], offer a bridge, facilitating compatibility among different WMS.

Complementing the TES, GA4GH also developed the Workflow Execution Service Schema (WES) [27], which defines an API to provide a consistent framework for developers to build compatible workflow execution systems. The WES specification includes endpoints for submitting workflows in accepted workflow languages, monitoring the execution progress, and retrieving the results. A WES implementation can leverage multiple workflow runners and support various workflows. The runners can then delegate the task execution to a common TES implementation. Many runners already implement execution support for most commonly used processing back-ends, like Slurm [28] clusters or Batch cloud services, to schedule tasks directly on these systems.

Overall, WMS, when employed according to the established best practices, simplify the process of running scientific algorithms in the cloud, providing researchers with instant access to virtually unlimited computing infrastructure and services without the hassle of provisioning and maintaining in-house clusters.

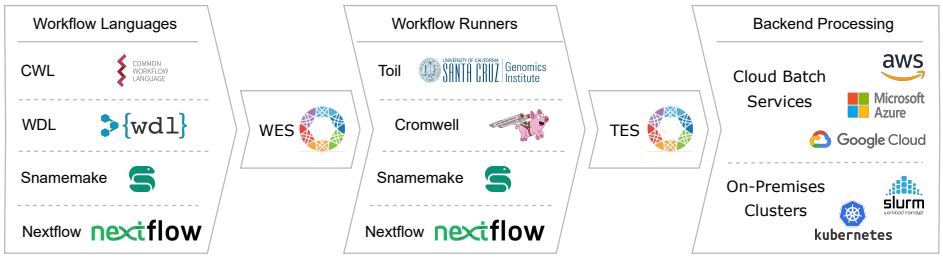

Fig. 1 Example of workflow languages, workflow runners and back-end processing connected through GA4GH WES and TES specification implementations

Despite these developments, resource allocation and monitoring in the Cloud environments pose a significant challenge, particularly since scientific tools may irregularly/unequally use assigned resources. This happens because each analytical task individually specifies resources needed in terms of the number of CPUs, memory, or disk space necessary for the execution. Thus, these values should be carefully selected to provide the task with enough resources for execution while avoiding overprovisioning resources that will not be used. A clear illustration of the consequences of resource mismanagement is evident in a June 2022 study conducted by Forrester Consulting for HashiCorp [29]. The study highlighted that a substantial percentage of cloud-related expenses incurred by companies were attributed to "idle or underused resources" (66%) and "overprovisioning of resources" (59%). This phenomenon, often called "cloud waste", underscores the financial implications of inefficient resource utilization.

The monitoring services offered by cloud providers display the use of resources in the cloud infrastructure at the virtual machine level or in terms of cluster reservation percentage. However, these services do not allow for the inspection of resource use for individual containers running alongside others in a cluster. To address this gap, a custom system is needed to extract information about how each container's assigned resources are used and determine how the resources should be adjusted for the tasks. This need also extends to creating container resource requirements profiles based on the task parameters and data.

2 Goals

Detailed monitoring is crucial for identifying and debugging issues during the container's execution, such as a lack of memory, CPU, or disk space. Additionally, by closely monitoring the resources a task uses, it is possible to adjust the resources assigned and optimize the execution costs, which is essential in a cloud environment where prices vary depending on the resources used. Furthermore, predictive models can be developed to forecast the costs of an analysis based on previous executions, enabling efficient scheduling of tasks with an awareness of the estimated duration and size of the resources needed for the execution [30]. In addition to the requirement for detailed monitoring of individual tasks, effective monitoring across diverse systems

and the necessity of unified metrics are equally important as researchers more frequently utilize multiple WMSs to run workflows from public repositories, which are often defined in different workflow languages.

In response to these challenges, we present a resource usage monitoring solution tailored for scientific WMS. We propose a flexible, event-driven architecture that works on multiple platforms and collects detailed and aggregated data across various WMS tasks. The system uses existing monitoring components and includes task labeling for efficient data organization. Built with current technologies, it is modular and integrates easily with other systems. Initially implemented on AWS, our solution is adaptable to various cloud or in-house platforms, demonstrating its wide applicability and scalability.

The Cloud Monitoring Kit (CMK) architecture is especially useful to researchers with a background in cloud systems development who are looking for a solution providing transparency in the use of cloud resources by scientific analysis workflows. CMK is designed to provide essential insights through its intuitive dashboards, which display individual and aggregated metrics relevant to job performance. For instance, developers can leverage these tools to monitor resource consumption and tweak system configurations during the development phase or integration of tools into the execution environment, thereby enhancing efficiency and performance. Meanwhile, operations staff benefit from the capabilities for continuous performance monitoring and troubleshooting, which are crucial for maintaining system reliability. The system offers scientists a robust analytical platform to scrutinize data, facilitating informed decisions regarding system configurations. This adaptability of the CMK makes it particularly valuable in settings where precise resource management and systematic optimization are essential.

After the introduction, the remainder of the paper is structured as follows. First, we conduct a survey of related scientific work and monitoring solutions currently employed in Workflow Management Systems (WMS) operating within cloud back-ends. This review provides an insightful overview of the existing landscape in resource management and monitoring technologies. We then look into the core design of our proposed monitoring system, focusing on its architecture and innovative features that distinguish it from existing solutions. Then, we present a detailed evaluation and analysis of the results obtained from implementing our proposed solution. This section showcases the effectiveness of our system in real-world scenarios, highlighting its impact on enhancing resource management in cloud-based bioinformatics workflows. The paper ends with a reflection on our findings and a discussion of future research directions. We explore how this monitoring solution can be leveraged to advance the field of bioinformatics, addressing current challenges and anticipating future needs in this rapidly evolving domain.

3 Related Work

Recent studies underscore the necessity of monitoring to adaptively scale resources and mitigate service disruptions. Integrating various tools to assess resource requirements from historical bioinformatics analysis metrics represents a significant advancement

Table 1 Task resource usage monitoring capabilities for some of the most commonly used WMS referenced in bioinformatics publications, as stated in their documentation

WMS/TES	Aggregated	Configurable	Fine-grained
Nextflow	✓	-	-
Galaxy	✓	✓	✓*
cwltool	✓	-	-
Toil	-	-	-
Cromwell	-	-	-
Snakemake	-	-	-
miniwdl	-	-	-
Funnel (TES)	-	-	-

in cloud resource management. Fahad et al. (2017) describe the critical importance of monitoring tools for optimizing cloud resource deployments [31]. This work is complemented by the work of Birje and Bulla (2020), who provide a comparative analysis of commercial and open-source monitoring solutions [32], and Tyryshkina et al. (2019) [30], who demonstrate the utility of regression and classification models in estimating runtimes and memory needs. A comprehensive survey by Righi et al. (2019) [33] on system monitoring, data prediction, and resource management details the relationship between monitoring metrics, resource scheduling, and AI-based load prediction algorithms and provides a cohesive view of the current landscape. In the context of bioinformatics, CWL workflows are commonly employed to create workflow pipelines for analysis; the introduction of CWL-metrics by Ohta et al. (2019) [34] offers a framework for analyzing workflow tasks resource requirements within CWL workflows. Scientific workflows consist of thousands of highly parallelized tasks executed in distributed environments; therefore, advanced methods for tracing and investigating performance metrics and task behavior are required. Bader et al. (2022) propose a monitoring framework to address this need while indicating a need for better integration between workflow and resource managers to improve metric exchange and scheduling decisions [35]. Together, these studies underscore the significance of advanced monitoring systems in cloud environments. They highlight various strategies for improving resource management, from adaptive scaling to selecting optimal cloud service instances. However, they also highlight an unmet need for a straightforward way to achieve efficient resource monitoring. To address this knowledge gap, the capacity to monitor and comprehensively assess resource requirements was integrated in CMK to aid in future resource allocation efficiency through detailed task metrics analysis across diverse workflow management systems in bioinformatics analysis workflow environments. In this way, the scientific contribution of CMK lies in its provision of a robust, flexible solution for bioinformatics researchers, enhancing both the efficiency and cost-effectiveness of cloud-based scientific computing for bioinformatics workflow executions.

The open-source WMS frameworks used in scientific computation can provide information regarding resource use of tasks executed inside a workflow; however, each has limitations regarding aggregation, configurability, and/or granularity. We summarize the documented capabilities of 7 WMS and 1 TES in Table 1 and provide a

detailed example in Section 5. In the table, "Aggregated" metrics refer to summaries of resources at a task/workflow level (e.g., maximum CPU/Memory usage or total CPU time) and are usually available at the end of the execution. "Configurable" refers to the option to configure which metrics should be recorded, to add more metrics (e.g., via plugins), or to set up an external system where to store the metrics; finally, "Fine-grained" refers to the ability to see continuous metrics in real-time throughout the execution of each task. The metrics are useful in different scenarios, as explained in the following sections.

The first framework analyzed, Nextflow, monitors CPU usage, memory consumption, and disk usage for each process in the pipeline. It can also monitor the usage of cluster-specific resources, such as the number of nodes and CPU cores allocated to the pipeline. The Nextflow monitoring data can be shown in a final execution report, but it does not allow the metrics to be exported to other systems (e.g., into a shared metrics database) besides the report file, or extending the monitoring functionality, nor does it include fine-grained metrics.

The second framework analyzed, Galaxy [24] uses plugins to collect and display metrics for task executions. Galaxy can be configured with the Telegraf [36] monitoring agent to read fine-grained metrics, being the most configurable in terms of monitoring from the reviewed frameworks.

The third framework, cwltool, is the reference implementation of the CWL standard. Although cwltool does not include task metrics out of the box, the project cwl-metrics [34] implements a system to capture aggregated task execution metrics following task conclusion and store them in Elasticsearch [37], a search engine over massive datasets, for later inspection. It does not include fine-grained metrics.

The reference GA4GH TES implementation, called Funnel, can run tasks on different back-ends and could be used by multiple WMS to delegate task execution; however, it does not yet consider task resource usage monitoring. A monitoring system applied to the TES service could be an excellent way to unify the monitoring of tasks arriving from different WMS frameworks into one TES implementation.

To our knowledge, task resource usage metrics are not reported from the Cromwell, Toil, Snakemake, or miniwdl frameworks. To summarize the findings of our assessment of the state of the art, most WMS frameworks do not provide task resource usage metrics, and most of those that do are not detailed enough and/or do not support customization.

There are several Cloud-specific commercial solutions available for monitoring of resource usage, including Dynatrace [38], Datadog [39], and InfluxData [40]. While these support tracing of requests across different distributed cloud systems and address the needs of specific application services, like databases or webservers, they do not comprehensively collect and associate metrics tailored for analyzing the use of resources during scientific workflow execution out of the box. General-purpose monitoring stacks are designed to monitor various metrics and platforms and are often used by tools, like cwl-metrics or the Galaxy WMS, to build the specific monitoring solution. Some well-known monitoring stacks are the ELK Stack (Elasticsearch, Logstash, Kibana) or the Prometheus/Grafana stack, which can be changed into a Telegraf/Time Series

Database/Grafana stack. Grafana [41] is an open-source analytics and interactive visualization web application. Telegraf, an open-source agent for data collection, is also part of the InfluxData stack and can be used in almost any environment where there is a need to collect and send metrics to a central location. As our goal was to provide a solution for monitoring of resource usage of workflow tasks, we investigated the resource utilization of workflow tasks being executed within software containers, which provided insights into how specific tasks use their assigned resources.

To address the apparent lack of resource monitoring and management capabilities of open-source WMS frameworks, our aim was to develop a task resource usage monitoring solution that can be used with diverse bioinformatics WMS. Designed to be an event-driven reference architecture for a multi-platform monitoring system, our proposed framework addresses the constraints of existing cloud monitoring systems while enabling the collection of detailed and aggregated data across different WMS tasks. The latter is achieved via task labeling, facilitating convenient grouping and metrics aggregation. Examples of labels include user ID, workflow ID, tool name, etc. Our comprehensive approach builds upon existing technology to provide new, previously unavailable benefits in terms of modularity and capacity to be integrated with other data processing or collection systems. We implemented our framework as a serverless platform on AWS; however, the same functional components can be adapted to other cloud providers or in-house platforms, highlighting the broad applicability and scalability of the system.

Scientific data analysis, particularly in bioinformatics, with many different tools, parameter configurations, and file sizes, requires special monitoring to optimize the use of resources. While some WMS include basic monitoring, to the best of our knowledge, no generic solution for monitoring scientific workflows exists. The novelty of CMK lies in this solution being WMS-independent and using labeling for generalization and integration with these WMS without actual changes in the code of the tools.

4 Architecture of the Monitoring System

The architecture proposed in this work comprises an advanced monitoring system, the Cloud Monitoring Kit (CMK), to collect, store, and access metrics of bioinformatics workflow tasks being executed in a container-based batch-processing environment. Even though the solution is applied here to bioinformatics, the same principles can be extended to other domains that use similar WMS. This monitoring system offers insights into resource use at a granular level for every task executed in the system, as well as metrics for grouped tasks based on labels that the user and administrator define in the system. It also summarizes the use of resources based on these defined groups. The architecture shown in Fig. 2 is defined via generic components that will be later instantiated to a particular implementation on a public Cloud provider. Therefore, the proposed architecture can be generalized to support multiple computing back-ends.

4.1 Proposed Architecture

A container-based batch-processing environment requires the deployment of a cluster, a set of virtualized computing nodes on which containers are executed with the help of

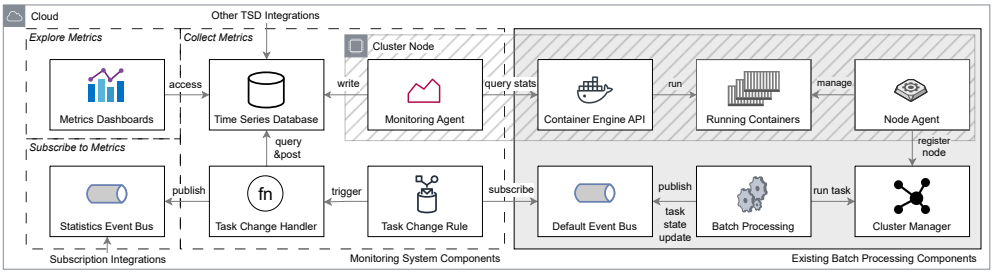

Fig. 2 Architecture of CMK defined with generic components. The gray area includes the components commonly found in container-based batch-processing systems. The white areas on the left include components required to gather, process, and visualize advanced task-level monitoring metrics

a Container Orchestration Platform such as Kubernetes [37], Nomad [42], or Amazon ECS [43], in the case of AWS Batch. At the core of the monitoring system is the monitoring agent, which runs as a service on each cluster node. The agent is responsible for connecting to the Container Engine API at the node level and collecting metrics regarding the use of available resources by each container running on the same node. The agent only needs to capture the metrics visible at the container level, and there are no requirements regarding the container image content for the monitoring to work. The collected metrics are saved into a Time Series Database (TSD) for future exploration. The TSD is a specialized database optimized for handling time-series data and has built-in functions and operators that allow for efficient data exploration. Each TSD entry contains the values of the resource use metrics, labeled with attributes such as the task and node IDs. Task attribute labeling is useful for creating detailed queries to the database and extracting statistics on data groups, such as task duration or resource usage statistics, arranged by task utility or by any custom label like user, project, etc. These aggregated metrics enable comparison of the statistical values between changes in bioinformatics tool versions or execution environment configurations. The TSD is available for direct querying, allowing the full range of collected metrics to be inspected and resource usage dashboards to be created.

TSD integrations like dashboards or REST APIs can be configured to query the database and show metrics and indicators in a management web, e.g., with a TES or WES front-end. Additionally, every time a task is completed, the batch system notifies a default event bus that a task-change event has occurred. A task-change rule captures these events and invokes a handler function, which queries the database to create aggregated statistics and publish them into the Statistics Event Bus. Other components can be integrated with this custom event bus to perform further processing or to store the aggregated statistics long-term for later use. The advantage of integrating a system with the Statistics Event Bus over the direct connection to the TSD is that the integration would receive the new events in real-time once the aggregation is performed. This can be used in examples like long-term storage, insertion into other systems, or further processing and addition of other computed metrics into the database. The Statistics Event Bus can be disabled if no other integrations are used.

Table 2 Equivalence of components between three cloud providers and open-source options

Component	AWS	Azure	Google	Open Source
Time Series Database	Amazon Timestream	Azure Time Series Insights	Cloud Bigtable	InfluxDB, Prometheus
Batch Processing	AWS Batch	Azure Batch	Batch	Kubernetes, Nomad
Function	AWS Lambda	Azure Functions	Cloud Functions	Open FaaS, OpenWhisk
Event Bus	Amazon EventBridge	Azure Service Bus	Pub/Sub	Kafka, RabbitMQ

The labeling of tasks can be done through container labels or environment variables. These key-value entries can be set either in the workflow or by configuration when the task is submitted. These labels can be collected via the monitoring agent. The different workflow language standards and alternative frameworks already allow or adopt this functionality. The TES specification, for instance, includes optional tags and environment variables that can be assigned to the submitted tasks. One important aspect to consider for the task labeling capabilities is that the submission system is responsible for forwarding those to the container attributes.

4.2 Implementation

A summary of the services, including three cloud providers (AWS, Azure, and Google Cloud) and open-source platforms, through which our proposed architecture can be implemented is shown in Table 2. For a proof-of-concept demonstration, we implemented our architecture on the AWS public cloud platform, as it is an established leader in cloud service offerings and is widely adopted in bioinformatics research. Moreover, because different scientific WMS support the AWS task scheduler (AWS Batch) for the execution of tasks [11, 19, 20], an AWS implementation represents an ideal use case for this monitoring system.

To align with the DevOps practice Infrastructure as Code (IaC), and to facilitate collaboration, version control, and deployment, the architecture was implemented using the AWS Cloud Development Kit (CDK). The CDK framework provides a practical solution for cloud infrastructure management, allowing the definition and provision of AWS resources, like databases and lambda functions using familiar programming languages. The CDK application is synthesized into an AWS CloudFormation [44] template, a lower-level declarative format that can be deployed in AWS. A diagram of the proposed architecture with AWS components is shown in Fig. 3.

In the AWS implementation, metrics are collected from tasks running in AWS Batch, which is configured with EC2 Compute Environments. These EC2 Compute Environments are a cluster of nodes that can scale automatically to match the resources required by the tasks in the queue.

As the proposed solution is designed to be scalable and handle large-scale, distributed batch task environments, the architecture is implemented on serverless

services—including the database, functions, and event bus—allowing the system to scale automatically in response to increased usage. In this case, serverless services refer to cloud computing models in which the cloud provider automatically manages the allocation and provisioning of servers. Additionally, the agent component of the system runs on each cluster node, so it is scalable with the number of nodes in the clusters, ensuring that the monitoring system can adapt to changing resource usage patterns.

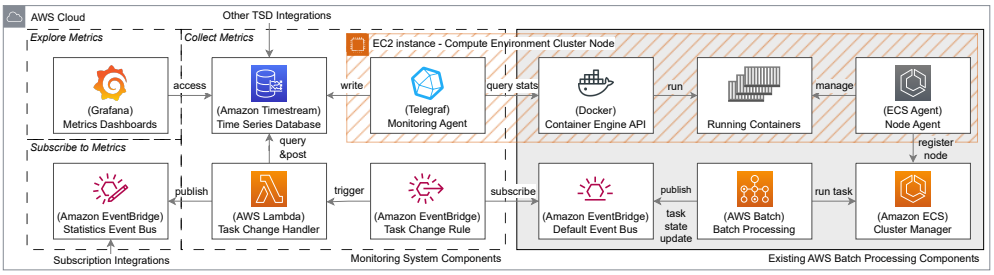

Fig. 3 Architecture diagram of the monitoring system implemented on AWS infrastructure, monitoring the AWS batch tasks. The gray area includes the components found in AWS Batch. The white areas on the left include the AWS components required to gather, process, and visualize advanced task-level monitoring metrics

For metrics collection, we used Telegraf, a production-ready, open-source, plugin-based server agent for collecting, transforming, and reporting metrics. Telegraf is widely used in many applications, including monitoring databases, systems, or IoT sensors; has a minimal memory footprint on the nodes; supports the collection of metrics from many systems, as well as options for data output that are configurable via plugins. Telegraf can fulfill many monitoring requirements through plugins, which can enable the capture of node metrics (e.g., system load, CPU power, or temperatures), which might be useful in some cases. For the AWS Cloud environment in our use case, the container metrics are collected using the Docker Input plugin, which connects to the internal Docker API of the host node via the *docker.socks* file to read the container statistics. The Telegraf agent is deployed as a Service on the ECS cluster corresponding to the AWS Batch Compute Environment, meaning the agent runs as a container in each node in parallel with the batch tasks.

Configured with the Telegraf agent for storage of the collected metrics is an output plugin for Amazon Timestream, a serverless service offered by AWS that eliminates tasks, like server provisioning and scaling, that are common in traditional relational databases. Timestream efficiently stores and retrieves time series data by offering unique built-in functions, such as time-based windowing and interpolation, which are not typically available in standard SQL databases. Because Timestream's pricing is usage-based, with charges derived from the amount of data written, queried, and stored, it is adaptable to fluctuating workloads. For new users seeking to test this service, a free tier is available for one month that allows testing within reasonable

limits. Together, these aspects make Timestream a practical choice for managing and analyzing time series data in the AWS implementation of our solution.

With regards to data access, there are multiple ways this can be achieved, depending on use case and integrations with other systems or components. Timestream, for example, allows data to be queried directly on the service website or via the service API using SQL-based queries. Alternatively, Grafana [41], an open-source monitoring and observation platform, provides interactive visualizations and can be integrated with Timestream, allowing the creation of dashboards that show Timestream data. In our reference architecture, we show how the Grafana dashboard can be used for direct visualization of resources used by a task or by a group of tasks (Fig. 7 and Fig. 8).

In addition to user metrics, the aggregated task statistics can be accessed at the time of creation by subscribing to a custom Statistics Event Bus, where task metrics are aggregated and published every time a task is completed. Components that subscribe to the Statistics Event Bus are configured internally in AWS, and the security and access are configured via internal system policies. To route events from various sources in our reference framework to their appropriate targets based on user-specified rules, we employ Amazon EventBridge, a managed service that supports event-driven architectures. Overall, this software architecture pattern enables different system components to communicate with each other through events without being directly connected, thus promoting decoupling and scalability. Other services like Amazon SNS [45] can be used either together with Amazon EventBridge, or replacing it, to achieve similar real-time event notification and decoupling of components, specially when working with integrations outside of AWS.

Regarding cost, in certain use cases, our monitoring system can be used free of charge for up to a month. While all services that comprise the monitoring system follow the serverless principles managed by AWS, meaning that the use-based cost structure considers the number of requests and/or the size of the data processed, some services, like AWS Lambda and Amazon EventBridge, have a free tier, permitting certain use cases to run on these services at no cost within specified usage boundaries. Amazon Timestream has a one-month free tier with boundaries that limit the size of the ingested and queried data besides the size of the stored metrics. The AWS Batch service is free, and only the EC2 instances are charged as part of the execution environment. Additionally, the Grafana dashboards we provide can be installed in the Grafana Cloud service, which provides a free account with up to three users forever. After the limitations to the free services are exceeded, the cost is proportional to the number of tasks, the time the monitoring system is active, and the size of the metrics processed and stored.

We show in the next section that CMK can be used as-is for different WMS with the same compute back-end. If we were to change components of this architecture, we would need to adjust the surrounding linked components to address the changes. To use another container engine for example, the Telegraf agent should be configured with another data input plugin that can read the task resources metrics. The metrics names can be overridden to match the current ones and to be able to reuse the same dashboards. Alternatively, the queries in the dashboards and the aggregation functions must be adjusted accordingly. Changing Amazon Timestream with another Time

Series Database, for example, would require adjusting the Telegraf configuration to write the metrics into that database, and the queries from the Grafana dashboards and aggregation function would need to be adjusted to translate any Timestream-specific functions or features to their equivalents in the target database SQL syntax dialect.

To use CMK in other cloud providers or in-house clusters, the same architecture would need to be re-implemented with the respective components. The use of CMK in a hybrid HPC-cloud environment to unify metrics would be possible as long as the local and cloud configurations concur in storing the same metrics. To simplify the architecture, all Telegraf agents from cloud and HPC should send the data to the same database. With the current implementation, adding access for the HPC to write metrics in Amazon Timestream is possible. Besides this, the Task Change Handler should be called from the HPC task manager to trigger the task summary metrics creation.

5 Evaluation

5.1 Introduction

This section evaluates the monitoring system using a commonly used bioinformatics workflow as a case study. To test the system functionality, we deployed the infrastructure in an AWS Batch compute environment, configuring two WMS (Nextflow and miniwdl) to run an example workflow for gene expression analysis. We first observed what metrics we could capture in addition to those reported by the cloud or the WMS. Then, we evaluated how the labeling options could enhance the utility of these aggregated metrics. In the following subsections, we describe the workflow implemented to evaluate our monitoring system to provide an example of a real-world context to which our system can be applied. Next, we provide details on the configuration of the AWS compute environment used for running the tasks, the configuration of the monitoring system, and the two WMSs used. Then, we explore our labeling setup for this use case, followed by a short description of the workflow and the data used. Finally, we observe the captured metrics, comparing them with what could be obtained from the WMS, and the cloud in the same run.

5.2 Workflow Description

To provide a use case for the evaluation of CMK, we used a published workflow for transcript expression quantification analysis [46]. This workflow, shown in Fig. 4, uses RNA sequencing (RNA-seq) data and a genome reference file as inputs to create a sequencing quality report and a transcript expression quantification table as output using three processing tools: 1.) The FastQC tool is executed independently for each set of paired sequencing reads to create the quality control report; 2.) the Salmon Index algorithm creates an index of the reference genome, which is used, along with the paired-reads dataset, in 3.) the Salmon Alignment Quantification step to create an alignment quantification.

We used example data from two NCBI projects to execute the selected workflow. The first project was NCBI BioProject PRJNA419302, comprised of 12 BioSamples for Monilinia laxa and one reference transcriptome assembly (TSA: Monilinia laxa, transcriptome shotgun assembly). The second project was the NCBI BioProject PRJNA325641 comprised of 12 BioSamples for Neisseria gonorrhoeae and a reference transcriptome (Neisseria gonorrhoeae FA 1090 [GCA 000006845]).

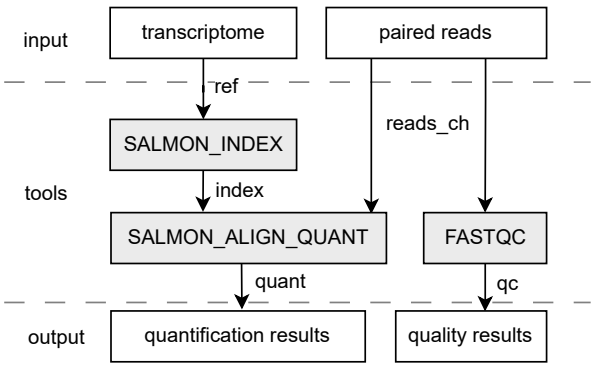

Fig. 4 A diagram of the evaluation workflow containing input and output files in white and the tools in gray

5.3 AWS Compute Environment

After implementing the workflow definition in the Nextflow and WDL languages and adding labels for grouping and aggregating metrics, the workflow tasks were executed on AWS Batch. We used the AWS Core Environment template, available in the Genomics Workflows on the AWS website [47], which configures AWS Batch as a base computing environment to use with WMS. The VM hardware setup in the template contains a range of CPU performance, balanced general-purpose, and memory-optimized hardware to cover different possible task requirements. This is a multi-node configuration where the environments automatically grow by adding new VMs as needed. Following AWS Batch setup, we configured the Nextflow v23.04.1 and miniwdl v1.11.0 workflow managers to send the tasks to the AWS Batch service. The full compute environment configuration is available in the project GitHub repository[1], and the schematic is shown in Fig. 5.

5.4 Monitoring Configuration

The monitoring system CDK cloud application was deployed in the same region and account as the AWS Compute Environment used for computation.

A configuration file specifies a list of parameters that set up the monitoring on the Computation Core Environment set up with the AWS template, e.g., the clusters and

[1] CMK - https://github.com/biobam/cmk

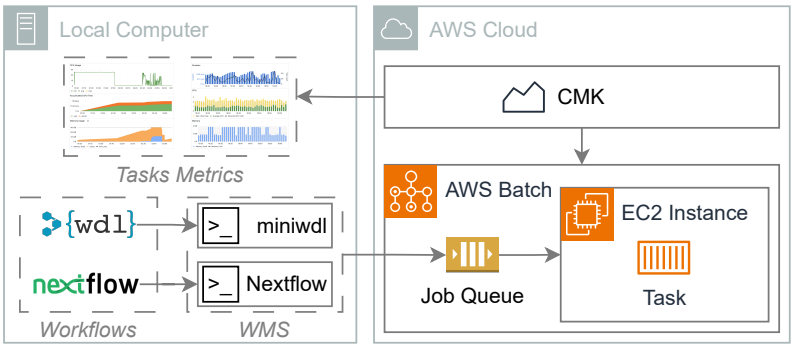

Fig. 5 Schema of the configuration for this evaluation. The two WMS running on a local computer were configured to communicate with AWS Batch to schedule tasks in a queue. The tasks execution running in the ECS cluster associated with this Batch Compute Environment is monitored and can be explored in the Grafana Dashboards

network IDs where the agent should be deployed as a service or database names and persistence duration boundaries.

5.4.1 Data Staging and Task Execution Structure

Data staging is the process by which the data to be analyzed is made available for a task. The data is commonly stored in an object storage service in cloud environments. Unless the analysis can stream the data to be processed, the data is copied locally for the duration of the execution, where the algorithm can access it directly via POSIX [48] standards. The contrary applies to the output results that are usually moved from the local storage to the long-term storage for persistence. Fig. 6 illustrates three patterns for data staging and task execution seen in different WMS and TES implementations: a) wrapper – the pre- and post-staging processes are executed inside the same container as the algorithm; b) sidecar – a controller task handles the staging of the data and runs the main algorithm task on the same node, providing the paths to the local data; c) individual tasks – different tasks perform the staging and can run on the same or different nodes, as the data is localized into cluster-shared storage that is accessible to all tasks. The data staging approach influences how the metrics are captured. As we measure the resources used by the container, the metrics of the wrapper option (a) contain the staging and task processes together as one task, while the sidecar (b) and individual tasks (c) options clearly separate the metrics by tasks. For this evaluation, we simplified the use case by having only one option, wrapper cases (a). Nextflow already schedules tasks with the wrapper pattern where a script passed to the task handles the data staging inside the same container execution. For miniwdl, which would have an individual-tasks structure by default, we compose the pre-and-post commands in the workflow to stage the data. This way, the metrics should display a similar execution pattern when the same tasks run regardless of the WMS.

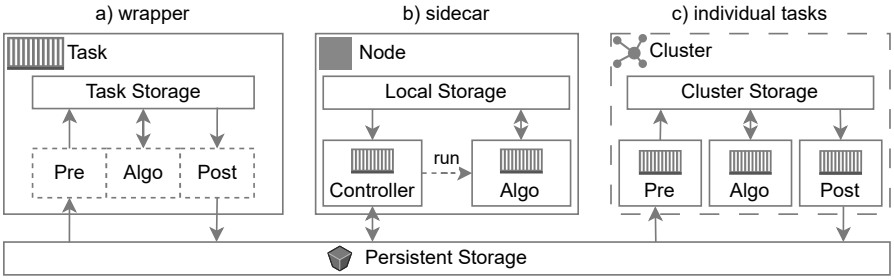

Fig. 6 Schema of three approaches for task data staging seen in different TES and WMS implementations

5.5 Labels

Labels are key-value pairs attached to the metrics when captured by the CMK. The label keys to be stored need to be listed in the configuration. The value of the labels can either come from the task execution environment, or it can be provided through the WMS when submitting a workflow. There is only one required label for the system to work, one that uniquely identifies each task. The property "taskIdLabel" in the CMK configuration specifies the key of this identifier. In our case, running on AWS Batch, this key is "AWS_BATCH_JOB_ID", and its value is provided by the service in the task environment. This label is used internally to create task-aggregated metrics. While additional labels are not mandatory, they enrich the exploration of resource usage. The labels act as partitions and are useful in grouping tasks and visualizing statistics of one task group that matches one or many labels. The same labels should be used across multi-platform or multi-WMS deployments. The tool code repository documents a list of labels that are often useful to capture in a scientific workflow configuration environment.

For this evaluation, we enable tracking of the following labels: a) "C_WMS" – the name of the WMS that scheduled the task; b) "C_TOOL" – the name of the software running in the container; and c) "C_DATASET" – the name representing the input data, which in most cases is the name of the input file. Additional labels of interest (e.g., user ID or workflow ID) can be easily added by listing them in the CMK configuration and by adding the corresponding labels to the workflow tasks when scheduling the workflow.

For the labeling part of the monitoring system, the WMS is required to forward these environment variables to the Docker container running the task, a process that is monitored by the agent that reads the labels. The original workflow has been modified to add these three custom labels to the tasks as environment variables. Nextflow allows the specification of 'container options' to set the values for the labels. For fixed parameters, like C_WMS, the value can be set globally in the configuration file for all tasks; for dynamic values, like C_DATASET, they must be set to change dynamically with each task execution, depending on the input file name. The standard WDL, on the other hand, is currently adding language support for assigning environment variables

to the tasks[2]. For now, this support is adopted by the miniwdl WMS, which we use in this evaluation, starting with software version 1.8.0 as an experimental feature.

5.5.1 Metrics Comparison

We provide a comparative analysis of the monitoring capabilities of our system (CMK), alongside existing solutions like CloudWatch and Nextflow, across key dimensions in Table 3. "Workflow Aggregated Metrics" offer a high-level view of workflow processes, such as total CPU hours and execution time. In contrast, "Task Aggregated Metrics" provide granular details on individual tasks, including metrics like average and peak CPU usage and total CPU time. Additionally, "Container-independent Ability to Capture Metrics" refers to the system's capacity to collect data without depending on container-specific elements, ensuring broader applicability. "Near Real-time Host Metrics" focus on the continuous monitoring of the host environment, tracking real-time performance and resource utilization, while "Near Real-time Task Metrics" concentrate on the detailed observation of each task's resource usage during execution, which is critical for identifying inefficiencies and bottlenecks. Finally "Custom Aggregated Metrics" enhance the system's versatility, allowing users to compile metrics based on specific labels for tailored analyses, such as aggregating data by user ID or tool. This comprehensive classification underlines the CMK system's strengths in providing detailed, real-time, and customizable monitoring solutions compared to its counterparts.

Table 3 Comparison of metrics that can be obtained from CMK, compared to the AWS-provided CloudWatch metrics and the metrics provided by the WMS

Description	CMK	CloudWatch	Nextflow
Workflow Aggregated Metrics	✓	-	✓
Task Aggregated Metrics	✓	-	✓
Container-independent ability to capture metrics	✓	✓	-
Near real-time host metrics	✓	✓	-
Near real-time task metrics	✓	-	-
Custom aggregated metrics	✓	-	-

In our case, the AWS cloud provider provides metrics on the infrastructure used, like metrics of the hosts, overall cluster metrics, or metrics grouped internally by the task-definition name. These metrics are useful for understanding how the overall resources are being used, but they do not allow to determine resource use at the task level.

At the end of workflow execution, Nextflow provides a report that contains workflow-aggregated metrics—like the total CPU hours used to execute all the tasks in the workflow—and displays plots of the distribution of resources, aggregated for each process name, from an overall workflow point of view. The HTML report generated post-execution also includes totals for each task, but it does not include real-time

[2]The approved pull request #504 of the specification repository adds support for environment variables (https://github.com/openwdl/wdl/pull/504)

task metrics or details of the use of resources during the execution. Moreover, the metrics are collected in the background of each tasks' execution, and are dependent on the availability of additional tools (awk, date, grep, ps, sed, tail, tee) inside each task container for the metrics collection to work. This means that "distroless" images that contain only the application and its runtime dependencies (i.e., one of the container best practices [49]), will not report metrics unless the image is wrapped in another container layer to add these dependencies. Since the metrics in CMK are collected at the container level with one agent running on each computation node, it has no container environment requirements and is compatible with any image. CMK collects all the metrics that can be viewed by workflow or in general for all tasks of some grouping. Going beyond the capabilities of Nextflow's grouping of a workflow view, our labeling methodology allows for more flexible statistics grouping and filtering, like grouping tasks by the container image, including all tasks from all workflows executed in a time window, or grouping tasks by any other custom labels, for example, to see the total CPU hours of one user ID. Additionally, our system is compatible with different WMS tools, and it adds monitoring capabilities to the ones that do not yet provide them, like miniwdl.

Two preconfigured dashboards are provided with CMK to explore the collected metrics, one at a task level with continuous metrics, showing the use of metrics throughout the execution time (Fig. 7), and one with aggregated metrics and charts grouped by the available labels (Fig. 8). Any configured labels can be selected in the aggregated dashboards to filter and group the tasks.

5.6 Evaluating CMK in an Industry Context: A Case Study

To assess the applicability of the proposed system in an industrial setting, we implemented CMK for workflow management at BioBam Bioinformatics (BioBam), a life science company that offers software solutions to accelerate genomics research. BioBam's software platform, OmicsBox [50] (formerly known as Blast2GO [51]), comprises a suite of tools that facilitate the analysis and interpretation of biological data. These tools are designed to make complex bioinformatics analyses accessible to a wider range of researchers, including those without extensive computational backgrounds. The company leverages AWS to accommodate the execution of bioinformatics algorithms, thereby supporting computational resource-intensive research in genomics, transcriptomics, and metagenomics.

Despite the many diverse bioinformatics algorithms executed by BioBam daily, the company did not have a way to monitor workflow resource use in detail. Therefore, it presented an optimal use case for evaluating the CMK architecture and testing its functionality. CMK allowed BioBam to conduct advanced monitoring to identify and mitigate the company's potential cloud waste. Implementing CMK into BioBam's setup illustrates its potential to enhance resource efficiency and reduce operational costs in cloud-based bioinformatics computing.

CMK was configured to run on the company's AWS cloud infrastructure, which is based on AWS Batch, to gather metrics of the tasks that arrive in the system. Information regarding the monitoring of resource use of over 70 cloud-executed bioinformatics algorithms at BioBam was collected over the course of approximately 6 months. The

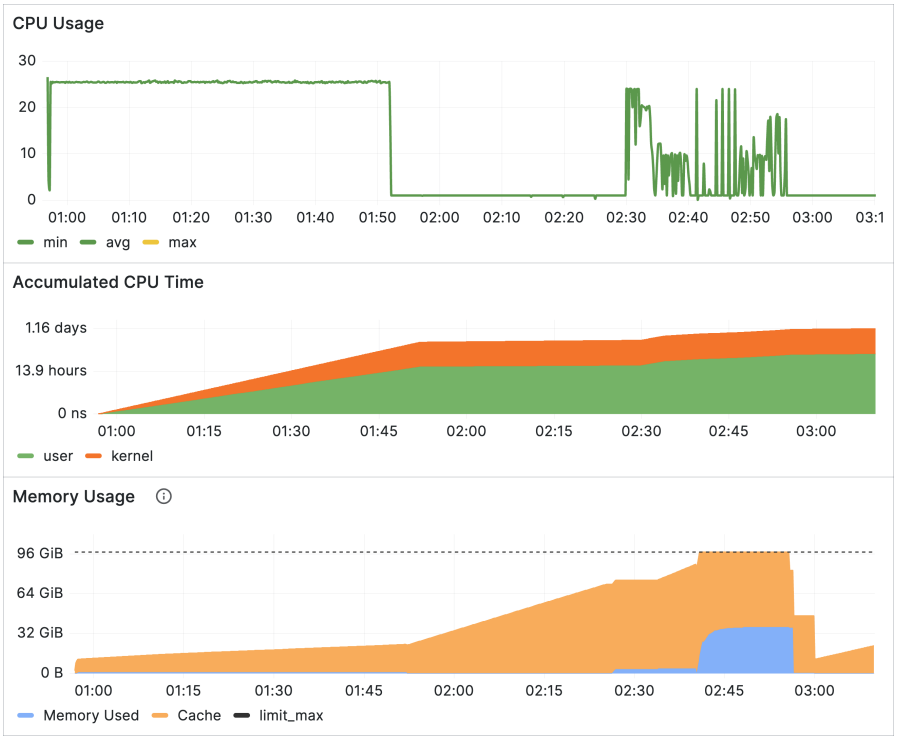

Fig. 7 Example detailed metrics of CPU usage, CPU time, and memory usage from the individual task dashboard

initial benefits of using the monitoring system were evident in deploying new algorithms. CMK assisted the developers in setting the container resource requirements by simplifying the process of benchmarking tools and providing data to recognize resource usage patterns. Another significant benefit was the help in diagnosing tasks that fail or abnormally extend beyond their expected completion times. By providing detailed analytics on resource usage and execution patterns, the system aided developers in identifying inefficiencies or resource bottlenecks that may cause these issues.

A "Cloud Waste" dashboard was created in Grafana to better understand resource usage efficiency. This dashboard aimed to measure the gap between resources allocated to tasks and their actual utilization, offering a comprehensive overview of all executed algorithms. Initially focusing on the most utilized algorithms and those consuming significant resources over extended periods, this analysis has led to notable improvements.

From the algorithms that produced the most cloud waste, the top three were cd-hit, hmmscan, and diamond. Cd-hit, used for clustering, initially had been configured to run with 24 vCPUs, and CMK detected a waste of 95% based on CPU usage. After seeing the average and peak CPUs, it was reconfigured to run with 8 CPUs, and the waste was lowered to 79%. Assigning fewer resources could cause the task to run longer but more efficiently if the tool itself cannot effectively use all the CPUs assigned. Another example is hmmscan, which performs Pfam search to predict coding regions

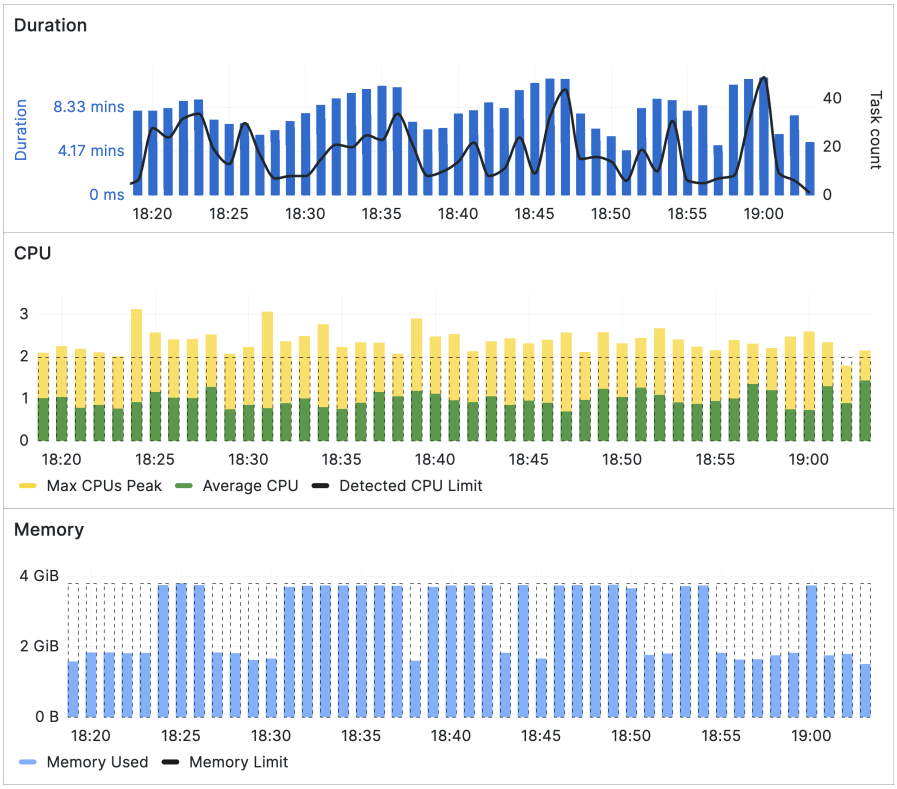

Fig. 8 Example of aggregated metrics exploring the use of resources filtering the "C_TOOL" label for a specific value, showing a histogram of the aggregated values during a time interval

in sequences. Initially configured with 6 vCPUs and 12 GB of memory, producing a cloud waste of 79% After seeing the patterns of CPU and memory for diamond, a tool for sequence alignment, clustering, and classification, two different CPU/memory configurations were created: one for large databases like NR (Non-Redundant Proteins), and the other for smaller databases like Swissprot. Additionally, the memory, even though not reserved directly, seems useful to cache the large sequence database files, with more memory allowing for faster read access.

As seen above, CMK provides an overview of the executions in the cloud that can focus on one tool or details of a tool, based on labels. Certain algorithms that consistently allocated more resources than necessary were identified and adjusted to match their needs more closely. Various configurations have been adopted for algorithms with a high variability of resource requirements influenced by parameters, e.g., the target query database, to optimize resource use. Long-running tasks characterized by fluctuating resource demands—periods of high or low CPU usage—were examined to find or develop alternative computational environments that could adapt to these changes more efficiently, potentially by migrating tasks or tailoring the environment to the task's current phase. These enhancements have yielded tangible benefits, directly contributing to cost savings by optimizing cloud resource usage and indirectly by

reducing developers' time configuring or troubleshooting algorithms. The monitoring and optimization of task configurations is an ongoing process in the company.

The modular structure of CMK facilitated the integration of an extension to compute and capture the cost of tasks running on spot VMs. Spot VMs allow users to bet on lower pricing for computing capacity, with the risk of resources being reclaimed by AWS at any time. This extension uses real-time data of the spot instance price and the proportion of resources allocated to the task, enabling more precise cost management.

Based on the initial success of the implementation of CMK at BioBam, further developments are planned to extend the system's capabilities, such as exploring the use of historical data to improve predictions on resource needs for future tasks. This approach aims to refine how resources are allocated to cloud tasks, enhancing efficiency and reducing costs, benefiting both the company and the end users.

5.7 Discussion

We gained several insights during the implementation of CMK. Firstly, deploying the Telegraf agent was straightforward for capturing container metrics due to its plugin-based architecture, which allowed easy configuration and container metric collection. Next, integrating the Time Series Database (TSD), in our case Amazon Timestream, was effective due to Timestream's serverless architecture and built-in functions for handling time-series data, which made it an ideal choice for storing and querying metrics. Additionally, setting up Grafana to visualize these metrics was quick and highly valuable, as the interactive dashboards facilitated easy exploration of the collected data. Finally, implementing a labeling system for tasks also proved beneficial, enabling flexible grouping and aggregation of metrics; this approach allowed categorization and analysis of resource usage based on various dimensions, such as workflow, tool, and dataset.

We also gained insights regarding potential challenges that users should be aware of during the implementation. Initially, node removal issues arose in the ECS cluster because the Telegraf agent ran continuously. This problem was resolved by configuring the agent to start only when other tasks were on the node. Additionally, managing the deployment through AWS CDK provided flexibility and reproducibility but added complexity; users unfamiliar with CDK might find it challenging to maintain and update the infrastructure. Furthermore, another drawback of the CMK is related to the measuring time interval. The default configuration of the Telegraf agent in CMK measures metrics every 10 seconds, which might not be suitable for all kinds of workflows. For instance, workflows with bursts of short tasks that only last a few seconds will not have detailed progress metrics captured; instead, some summary metrics are available after the tasks terminate and are captured before the container instance is removed from the system. Lastly, defining meaningful custom aggregated metrics can be difficult due to the diversity of options and the specificity required to successfully align these with the research goals and operational needs.

As an easily accessible starting point, researchers and organizations can take the CMK system and deploy it as-is on top of the AWS Batch compute environment. With minimal configuration of labels and using the provided dashboards, users can explore the metrics and gain insights into their resource usage. The initial setup of CMK

provides significant value by offering detailed monitoring capabilities with minimal effort. Using visualization tools like Grafana is crucial for exploring and understanding the collected metrics, and setting up additional meaningful dashboards based on clear monitoring objectives like reducing cloud waste. Monitoring should be seen as an ongoing process, with regular reviews of collected data, configuration adjustments, and task setting optimizations based on observed patterns. This iterative approach will lead to continuous improvements in resource management.

6 Conclusion and Future Works

We have introduced a reference event-driven cloud-native architecture capable of enhanced resource utilization monitoring across multiple WMS in cloud computing environments. This architecture is designed to aggregate and analyze metrics from containerized tasks scheduled by various workflow systems, effectively addressing the critical need for understanding resource efficiency and cost optimization in large-scale bioinformatics workflows. The architecture leverages task labeling methods, enabling the grouping and aggregation of metrics in a WMS-agnostic manner. This approach simplifies the monitoring process and enhances the system's versatility. The system's modular design ensures easy integration with existing data processing and collection systems, offering a flexible and scalable solution for diverse cloud computing back-ends.

A reference implementation of this architecture for AWS, named CMK, has been created using the CDK framework; it has been released as open-source to support tasks running on AWS Batch Compute Environments and to serve as a foundation for future developments. Additionally, this approach aligns well with the principles of IaC, promoting best practices in collaboration, version control, and deployment in cloud environments.

The implementation of our system has been evaluated by running a common bioinformatics workflow from two different WMS, where we were able to monitor the resource usage of the tasks running in the workflows and to explore the resulting metrics in the provided Grafana dashboards. Additionally, the results of applying this system to an industrial setting (i.e., BioBam's computing environment) highlight the benefits of having the transparency and additional insights that the monitoring system provides. These insights can lead to cost reduction and optimizations in a setting where thousands of bioinformatics jobs are scheduled daily.

First, the collected metrics can be used to optimize the use of resources in the cloud to ensure that tasks reserve adequate resources. Secondly, tasks can be fine-tuned to better use those resources by adjusting internal threads or flows, resulting in faster analysis and decreased processing times. Thirdly, different profiles can be created to assign increased or reduced processing power to a task increase or decrease parallelization based on inputs, and reduce the time to results. Finally, the full modularity and flexibility of the monitoring system—to add data size metrics or cloud infrastructure cost metrics per task through task termination events and to add additional metrics through configuring plugins with Telegraf—allows users to address their unique data analysis needs.

Moving forward, integrating our framework with a greater variety of WMS and processing backends will showcase the broad application of the proposed architecture; researchers and developers who work with diverse WMS will find this compatibility invaluable for monitoring and optimizing their workflows. We aim to simplify the deployment process by making it available as parametrized CloudFormation templates. Each cloud provider has its own alternative to CloudFormation. OpenTofu [52] would be an open-source alternative for IaC, compatible with multiple clouds. The current CDK application might be harder to keep up with for users not familiar with the framework. Exploration into the use of advanced predictive analytics and machine learning using the data generated by the proposed architecture can further aid researchers in optimizing their cloud resource usage, potentially leading to substantial cost savings and improved efficiency in handling large-scale data-processing tasks. An application for this is estimating more precisely the real needs of tasks before execution. Incorporating checkpoint/restore-related applications could enhance the management of long-running tasks. This feature would be particularly advantageous in fields with intensive computational tasks, providing means to maintain progress and manage resources more effectively. By monitoring metrics like CPU throttling, memory swapping, and out-of-memory/disk errors, the system could detect whether the adjustment is too tight and can up-scale the executing node or migrate the task to an environment with more resources.

Acknowledgements. This work has received funding from the Valencian Innovation Agency (AVI) file INNTA3/2021/5 (INNODOCTO). GM and RN would like to thank Grant PID2020-113126RB-I00 funded by MICIU/AEI/10.13039/501100011033. This work was supported by the project "An interdisciplinary Digital Twin Engine for science" (interTwin) that has received funding from the European Union's Horizon Europe Programme under Grant 101058386.

We thank BioBam for supporting this collaboration, in particular E. Presa Díez and M. Benegas Coll for suggesting and providing links to suitable datasets for the workflow evaluation. Additionally, we would like to thank S. Hewitt for providing writing and editing assistance.

Declarations

- Funding
  This work has received funding from the Valencian Innovation Agency (AVI) file INNTA3/2021/5 (INNODOCTO). GM and RN would like to thank Grant PID2020-113126RB-I00 funded by MICIU/AEI/10.13039/501100011033. This work was supported by the project "An interdisciplinary Digital Twin Engine for science" (interTwin) that has received funding from the European Union's Horizon Europe Programme under Grant 101058386.
- Conflict of interest/Competing interests
  The authors have no relevant financial or non-financial interests to disclose.
- Ethics approval and consent to participate
  Not applicable

- Consent for publication
  Not applicable
- Data availability
  Not applicable
- Materials availability
  Not applicable
- Code availability
  The CMK platform and configurations tested were made available to the community as an open-source development in GitHub at https://github.com/biobam/cmk
- Author contribution
  R.N. conceptualized the project and methodology, developed the platform, and wrote the manuscript. S.G. and G.M. provided project oversight and guidance and acquired the funding required to perform the work. All authors reviewed the manuscript.

References

[1] Amazon Web Services (AWS) (2023). https://aws.amazon.com/

[2] Google Cloud (2023). https://cloud.google.com/

[3] Microsoft Azure (2023). https://azure.microsoft.com/

[4] Siddiqui, T., Siddiqui, S.A., Khan, N.A.: Comprehensive Analysis of Container Technology. 2019 4th International Conference on Information Systems and Computer Networks, ISCON 2019, 218–223 (2019) https://doi.org/10.1109/ISCON47742.2019.9036238

[5] Hale, J.S., Li, L., Richardson, C.N., Wells, G.N.: Containers for Portable, Productive, and Performant Scientific Computing. Computing in Science & Engineering **19**(6), 40–50 (2017) https://doi.org/10.1109/MCSE.2017.2421459

[6] Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. ISPASS 2015 - IEEE International Symposium on Performance Analysis of Systems and Software, 171–172 (2015) https://doi.org/10.1109/ISPASS.2015.7095802

[7] Giorgi, F.M., Ceraolo, C., Mercatelli, D.: The R Language: An Engine for Bioinformatics and Data Science. Life (Basel, Switzerland) **12**(5) (2022) https://doi.org/10.3390/LIFE12050648

[8] Fourment, M., Gillings, M.R.: A comparison of common programming languages used in bioinformatics. BMC Bioinformatics **9**(1), 1–9 (2008) https://doi.org/10.1186/1471-2105-9-82/TABLES/1

[9] Baker, M., Penny, D.: Is there a reproducibility crisis? Nature **533**(7604), 452–454 (2016) https://doi.org/10.1038/533452A

[10] Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., Stojanovic, L.: Common Workflow Language, v1.0. Figshare (2016) https://doi.org/10.6084/M9.FIGSHARE.3115156

[11] Voss, K., Auwera, G.V.d., Gentry, J., Voss, K., Auwera, G., Gentry, J.: Full-stack genomics pipelining with GATK4 + WDL + Cromwell. ISCB Comm J **6** (2017) https://doi.org/10.7490/F1000RESEARCH.1114634.1

[12] Goble, C., Cohen-Boulakia, S., Soiland-Reyes, S., Garijo, D., Gil, Y., Crusoe, M.R., Peters, K., Schober, D.: FAIR Computational Workflows. Data Intelligence **2**(1-2), 108–121 (2020) https://doi.org/10.1162/DINT_A_00033

[13] Herschel, M., Diestelkämper, R., Ben Lahmar, H.: A survey on provenance: What for? What form? What from? VLDB Journal **26**(6), 881–906 (2017) https://doi.org/10.1007/S00778-017-0486-1

[14] Khan, F.Z., Soiland-Reyes, S., Sinnott, R.O., Lonie, A., Goble, C., Crusoe, M.R.: Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. GigaScience **8**(11), 1–27 (2019) https://doi.org/10.1093/GIGASCIENCE/GIZ095

[15] Missier, P., Belhajjame, K., Cheney, J.: The W3C PROV family of specifications for modelling provenance metadata. ACM International Conference Proceeding Series, 773–776 (2013) https://doi.org/10.1145/2452376.2452478

[16] O'Connor, B.D., Yuen, D., Chung, V., Duncan, A.G., Liu, X.K., Patricia, J., Paten, B., Stein, L., Ferretti, V.: The Dockstore: enabling modular, community-focused sharing of Docker-based genomics tools and workflows. F1000Research 2017 6:52 **6**, 52 (2017) https://doi.org/10.12688/f1000research.10137.1

[17] Goble, C., Soiland-Reyes, S., Bacall, F., Owen, S., Williams, A., Eguinoa, I., Droesbeke, B., Leo, S., Pireddu, L., Rodríguez-Navas, L., Fernández, J.M., Capella-Gutierrez, S., Ménager, H., Grüning, B., Serrano-Solano, B., Ewels, P., Coppens, F.: Implementing FAIR Digital Objects in the EOSC-Life Workflow Collaboratory (2021). https://doi.org/10.5281/ZENODO.4605654 . https://zenodo.org/record/4605654

[18] Vivian, J., Rao, A.A., Nothaft, F.A., Ketchum, C., Armstrong, J., Novak, A., Pfeil, J., Narkizian, J., Deran, A.D., Musselman-Brown, A., Schmidt, H., Amstutz, P., Craft, B., Goldman, M., Rosenbloom, K., Cline, M., O'Connor, B., Hanna, M., Birger, C., Kent, W.J., Patterson, D.A., Joseph, A.D., Zhu, J., Zaranek, S., Getz, G., Haussler, D., Paten, B.: Toil enables reproducible, open source, big biomedical data analyses. Nature Publishing Group (2017).

https://doi.org/10.1038/nbt.3772

[19] chanzuckerberg/miniwdl: Workflow Description Language developer tools & local runner (2023). https://github.com/chanzuckerberg/miniwdl

[20] DI Tommaso, P., Chatzou, M., Floden, E.W., Barja, P.P., Palumbo, E., Notredame, C.: Nextflow enables reproducible computational workflows. Nature Biotechnology **35**(4), 316–319 (2017) https://doi.org/10.1038/NBT.3820

[21] AWS Batch (2023). https://aws.amazon.com/batch/

[22] Azure Batch (2023). https://azure.microsoft.com/en-us/products/batch/

[23] Google Batch (2023). https://cloud.google.com/batch/

[24] Giardine, B., Riemer, C., Hardison, R.C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W.J., Nekrutenko, A.: Galaxy: A platform for interactive large-scale genome analysis. Genome Research **15**(10), 1451–1455 (2005) https://doi.org/10.1101/gr.4086505

[25] TES specification (2023). https://github.com/ga4gh/task-execution-schemas

[26] Funnel (2023). https://ohsu-comp-bio.github.io/funnel/

[27] WES Specification (2023). https://github.com/ga4gh/workflow-execution-service-schemas

[28] Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **2862**, 44–60 (2003) https://doi.org/10.1007/10968987_3

[29] HashiCorp State of Cloud Strategy Survey (2022). https://www.hashicorp.com/state-of-the-cloud

[30] Tyryshkina, A., Coraor, N., Nekrutenko, A.: Predicting runtimes of bioinformatics tools based on historical data: five years of Galaxy usage. Bioinformatics **35**(18), 3453–3460 (2019) https://doi.org/10.1093/BIOINFORMATICS/BTZ054

[31] Fahad, A.M., Ahmed, A.A., Kahar, M.N.M.: The importance of monitoring cloud computing: An intensive review. IEEE Region 10 Annual International Conference, Proceedings/TENCON **2017-December**, 2858–2863 (2017) https://doi.org/10.1109/TENCON.2017.8228349

[32] Birje, M.N., Bulla, C.: Commercial and Open Source Cloud Monitoring Tools: A Review. Learning and Analytics in Intelligent Systems **3**, 480–490 (2020) https://doi.org/10.1007/978-3-030-24322-7{_}59/FIGURE

[33] Rosa Righi, R., Lehmann, M., Gomes, M.M., Nobre, J.C., Costa, C.A., Rigo, S.J., Lena, M., Mohr, R.F., Oliveira, L.R.B.: A Survey on Global Management View: Toward Combining System Monitoring, Resource Management, and Load Prediction. Journal of Grid Computing **17**(3), 473–502 (2019) https://doi.org/10.1007/S10723-018-09471-X/METRICS

[34] Ohta, T., Tanjo, T., Ogasawara, O.: Accumulating computational resource usage of genomic data analysis workflow to optimize cloud computing instance selection. GigaScience **8**(4), 1–11 (2019) https://doi.org/10.1093/GIGASCIENCE/GIZ052

[35] Bader, J., Witzke, J., Becker, S., Loser, A., Lehmann, F., Doehler, L., Vu, A.D., Kao, O.: Towards Advanced Monitoring for Scientific Workflows. Proceedings - 2022 IEEE International Conference on Big Data, Big Data 2022, 2709–2715 (2022) https://doi.org/10.1109/BIGDATA55660.2022.10020864

[36] Telegraf — InfluxData (2024). https://influxdata.com/telegraf

[37] Elasticsearch: The Official Distributed Search & Analytics Engine — Elastic (2024). https://www.elastic.co/elasticsearch

[38] Cloud monitoring — Dynatrace (2023). https://www.dynatrace.com/platform/cloud-monitoring/

[39] Cloud Monitoring as a Service — Datadog (2023). https://www.datadoghq.com/

[40] InfluxDB Cloud — InfluxData (2023). https://www.influxdata.com/products/influxdb-cloud/

[41] Grafana: The open observability platform — Grafana Labs (2024). https://grafana.com/

[42] Nomad by HashiCorp (2024). https://www.nomadproject.io/

[43] Fully Managed Container Solution – Amazon Elastic Container Service (Amazon ECS) - Amazon Web Services (2024). https://aws.amazon.com/ecs/

[44] Infrastructure As Code Provisioning Tool - AWS CloudFormation - AWS (2024). https://aws.amazon.com/cloudformation/

[45] What is Amazon SNS? - Amazon Simple Notification Service (2024). https://docs.aws.amazon.com/sns/latest/dg/welcome.html

[46] Wratten, L., Wilm, A., Göke, J.: Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. Nature Methods 2021 18:10 **18**(10), 1161–1168 (2021) https://doi.org/10.1038/s41592-021-01254-9

[47] Genomics Workflows on AWS (2023). https://docs.opendata.aws/genomics-workflows/quick-start.html

[48] IEEE SA - IEEE 1003.1-2001 (POSIX) (2021). https://standards.ieee.org/ieee/1003.1/1389/

[49] Bage, A.P., Saxena, S., Singh, Y.: A Brief Review on Lightweight Practice of Docker Vulnerabilities. Software Engineering Approaches to Enable Digital Transformation Technologies, 18–24 (2023) https://doi.org/10.1201/9781003441601-2

[50] OmicsBox – Bioinformatics Made Easy, BioBam Bioinformatics (2023). https://www.biobam.com/omicsbox/

[51] Götz, S., García-Gómez, J.M., Terol, J., Williams, T.D., Nagaraj, S.H., Nueda, M.J., Robles, M., Talón, M., Dopazo, J., Conesa, A.: High-throughput functional annotation and data mining with the Blast2GO suite. Nucleic Acids Research **36**(10), 3420–3435 (2008) https://doi.org/10.1093/NAR/GKN176

[52] OpenTofu (2024). https://opentofu.org/