Event2Ledger: Container traceability using Docker Swarm and consortium Hyperledger blockchain

Marco A. Marques[1], **Marcos A. Simplicio Jr.**[1], **Charles C. Miers**[2]

[1]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

[2]Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brazil

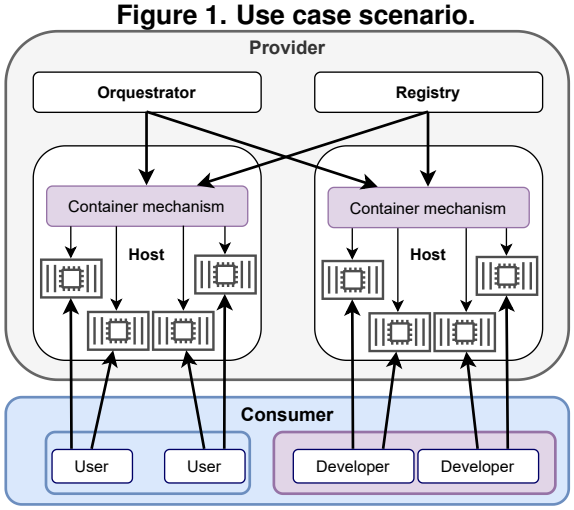
Abstract. *Cloud computing employing container-based virtualization enables dynamic allocation of computational resources, providing scalability and fault tolerance, etc. However, this flexibility could imply a drawback: container environment monitoring is highly challenging due to the large flow of calls and (de)allocations. In this work, we present event2ledger, a blockchain-based solution that implements a distributed log with data sent by authorized and customized collectors to a permissioned consortium blockchain, responsible for ordering and storage in a distributed and auditable manner. A proof-of-concept is implemented with a Hyperledger Fabric consortium blockchain, composed and maintained by the scenario actors (i.e., Providers, Users, and Developers), which stores signed container life cycle events.*

1. Introduction

Cloud computing applications usually are designed using a micro-services architecture where instead of a unique monolith file containing all functions and services, are several small ephemeral components (called micro-services) executed on demand. This characteristic offers many benefits to cloud applications, such as scalability and availability, but monitoring arises as a challenge, as these environments are composed of many applications with hundreds of micro-services, producing a high volume of data. In this context, monitoring consists of observing the execution of the virtual environment, collecting, and making available for analysis, periodically, a set of predefined variables.

Usually, cloud computing providers offer a monitoring system to the consumers, but they can also deploy an independent monitoring service [Dawadi et al. 2017]. This possibility brings autonomy to the actors, allowing the design and development of new monitoring solutions adapted to their needs. However, when there are different concurrent monitoring solutions to the same environment, disagreements regarding the collected information could happen, leading to different analyzes of the same scenario. Centralized monitoring solutions, by its side, depend on trusting to one actor all responsibility for not only the integrity but also availability and confidentiality of the collected data. Figure 1 represents the proposed scenario in which a distributed application, hosted on a cloud computing provider, is offered by the developer to a customer. The presented scenario shows the interaction between two main groups of actors, called Providers and Consumers. The Consumer group is composed of the developers, who host their applications at the cloud computing provider, and the users of these applications.

Figure 1. Use case scenario.



Each actor has different perspectives and needs, and the monitoring data is critical in some decision-making processes. In this scenario, their actions depended on the monitoring solution reliability, as its data are important input metrics for billing, performance, and usage analysis. Considering the proposed scenario, the problem presented by this work is: how to guarantee the validation and consensus among the actors participating in container virtualization environments about the order of container life cycle events and its data, allowing to audit them by any of the parties, and how to maintain this data preserving its integrity, authenticity, availability, and traceability.

This work is organized as follows: Section 2 presents the generic proposal solution to the problem. Section 3 presents the proposed architecture model adopted by Event2ledger, and details about the component integration. Section 4 details the minimum computational resources and prerequisites needed to execute the demonstration and characteristics about the blockchain and collector implementation. This section also brings information about the public repository containing the necessary files to execute the presented demonstration. Then, Section 5 discusses details about the solution implementation and execution, while Section 6 presents the conclusions about event2ledger implementation and how it can be adapted to several different scenarios.

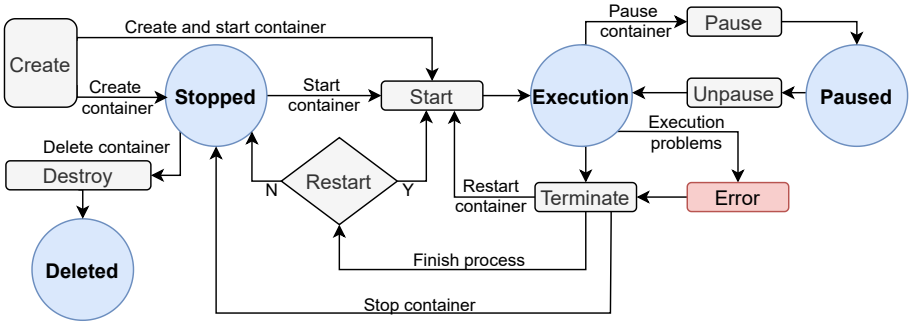
2. Proposed Solution

This work presents a distributed monitoring solution called event2ledger, which allows the joint and consensual actors to participate in container event validation, ordering, and storage processes. In the proposed solution, each cluster host executes an instance of the even2ledger collector. This collector connects directly to the local Docker *daemon* events endpoint and starts to listen for new income events. For each event in that host is generated a JSON formatted event register that is used by the collector to populate a new transaction proposal. Then this proposal is signed by the collector and sent to a Hyperledger Fabric consortium blockchain that performs distributed ordering, validation and storage, according to the policies defined in blockchain creation.

Thus, considering the proposed proof of concept scenario presented in Figure 1, the Provider is responsible for the cluster hosts' execution and maintenance, the container mechanism and orchestrator, and other infrastructure components necessary to execute the environment. The Developer, on the other hand, is responsible for creating and de-

ploying applications that are available for User execution on the cluster hosts. The User, by its side, executes the Developer's applications in the Provider's environment. From the Event2ledger point of view, the distributed consensus mechanism and the distributed ledger responsible for the storage of collected data are composed of a set of hosts maintained by all the actors (as presented in Section 3) that, following predefined security policies, grant the distributed model of the proposed solution. Typically Docker container life cycle composition has four states: Execution, Paused, Stopped, or Deleted. The transition between these states generates events that are the objective of the presented collector process. Figure 2 depicts the possible transitions between the life cycle phases of Docker containers.

Figure 2. Container life cycle. Adapted from [Mouat 2015].

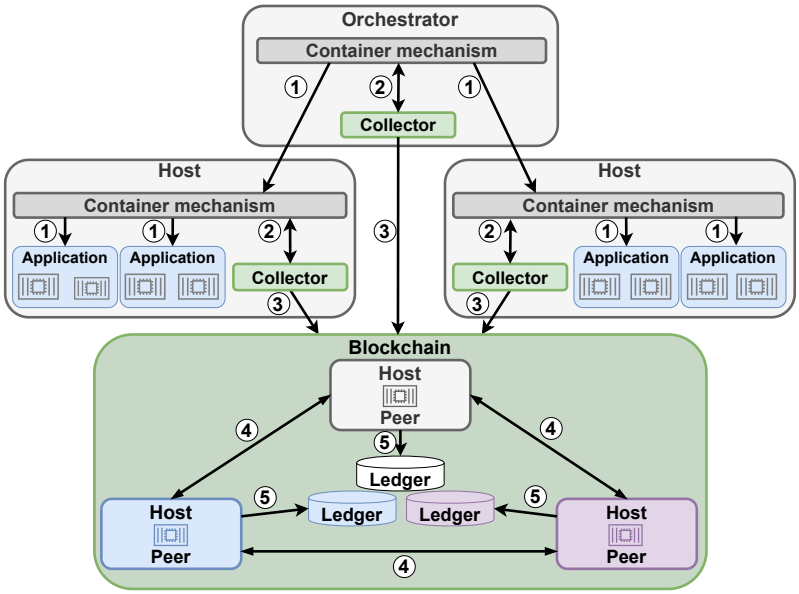


The transition actions between the life cycle stages can be performed by user action through a *Command Line Interface* (CLI) or using an orchestrator, with the Docker mechanism being the component responsible for managing the life cycle of the containers. Each object managed by Docker has a specific set of events related to its life cycle. Among the events available for the container object, this work focuses on the collection of creation, pause, finalization, and destruction events, specifically: create, destroy, pause, unpause, start, and stop. The proof of concept aims to demonstrate how ease is to adapt the collectors and chaincode, allowing the collection of any data regarding events, logs, and other similar order-based needs.

3. Architecture

The generic solution model, presented in Figure 3, details how the application architecture is deployed and executed in a proof of concept environment. The implementation environment considers a Docker cluster composed of three hosts, one as the orchestration manager and the other responsible for the application container execution that generates the events (1). Each host runs a collector that connects directly to the local container mechanism, receiving generated events in real time and in a non-intrusive way (2). To guarantee the collector's authenticity, asymmetric encryption is employed. For this, the collector that receives an event creates a transaction containing the event data, the chaincode function to be executed, and the necessary parameters, signing this transaction with its private key validated in a future step of blockchain transaction architecture. After generated and signed, the collector sends the transaction to the blockchain endorsing nodes (3). When receiving the transaction, these nodes execute the called chaincode function signing the results, but the blockchain world-state is not updated yet. The executed transactions are forwarded to a consensus mechanism that allows the actors to agree on their order (4). Finally, the events are validated and stored in a distributed repository (5) composed of three instances belonging to the actors.

Figure 3. Generic solution model.



The fact that each actor has a complete copy of the repository, together with the cryptographic chaining mechanism and the permissions policies applied to the blockchain, allows the auditing and tracking of changes to the stored data, improving the integrity, availability, and auditing characteristics. The proposed model allows high flexibility due to the possibility of creating customized collectors and chaincode functions according to actors' needs. Also, in agreement with a predefined set of policies defined in the blockchain configuration, actors can update the existing chaincode functions or discard old data that may not be useful anymore.

4. Demonstration & Implementation

The Event2ledger demonstration in this work implements the proof of concept scenario presented in Figure 3. This demonstration aims to implement a Hyperledger Fabric blockchain in a Docker environment where collectors connect to container event sources sending them to Event2ledger. With the environment and Event2ledger up and running, two test scenarios are performed, to demonstrate how the events are generated on the Docker side being collected, ordered, and stored in the blockchain nodes.

Considering that the event2ledger collector instances running in the cluster nodes execute the same way, this demonstration focus on the manager node instance that runs all solution components. This implementation can be replicated in other nodes from the same environment or used as a reference for different and new models. The available repository includes the necessary files to run the proposed Hyperledger Fabric blockchain (as detailed in Section 4.2), and the collector model presented in 4.3, which will connect to the local Docker *daemon* and start generating transactions as Docker events are collected. The event2ledger documentation is available at the Github repository[1], containing the prerequisites and the tutorial to reproduce this demonstration and also to implement it in all Docker Swarm cluster nodes.

[1]https://github.com/marques-ma/e2l_v0.5/blob/main/README.md

4.1. Environment

The proof of concept is implemented in a virtualization environment using Docker container and Docker Swarm orchestrator [Docker 2022]. The cluster that composes the Docker Swarm has, in total, three nodes: one manager and two workers. As all blockchain nodes are implemented in Docker containers and executed in the Docker Swarm manager node, this node represents the minimum hardware requirement for deployment and execution of this demonstration, allowing the event flow generation, collection, and storage. Table 1 presents the computational resources allocated to each node, considering the manager as the minimum requirement and all the nodes for the complete installation.

Table 1. Computational resources allocation in cluster mode.

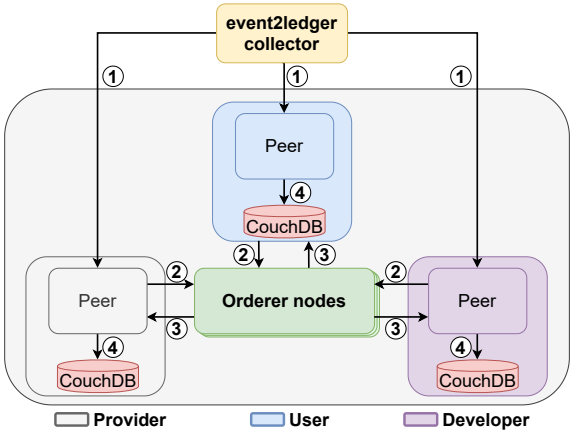
Node	vCPU	vRAM	Disk
Manager	2	8 GB	60 GB
Worker 1	2	2 GB	20 GB
Worker 2	2	2 GB	20 GB

The computational resources proposed in Table 1 consider the execution of the complete proposed infrastructure, including an orchestrator (*i.e.*, Docker Swarm), blockchain peers and orderer nodes, event2ledger collectors, and any other containers, applications, and services to support the proof of concept execution and simulation of events for collection and storage. All three cluster nodes run, as a base, a GNU/Linux Debian 11. For demonstration purposes, this work implements only the manager node, which runs a Docker instance and all the blockchain, reducing to the minimum the total resource needed to execute the complete flow of event generation, collection, and storage.

4.2. Hyperledger Fabric Blockchain

For ordering, validation, and storage of collected events, event2ledger uses a Hyperledger Fabric consortium blockchain version 2.2, in which nodes are containers created and executed in the Docker environment to be monitored, running specifically in the Docker Swarm manager node (Figure 3). The prerequisites necessary to install the blockchain, its API and execute the chaincode are listed in [Hyperledger 2022b]. With these requirements met, is possible to deploy a chaincode and perform all the cryptographic, ordering, and storing functions necessary to execute the proof of concept.

Figure 4. Proposed blockchain transaction flow.



The proposed blockchain model, presented in Figure 4, represents the blockchain components and the main steps performed after an event collection by any collector instance. When the collector receives an event from Docker, it generates a transaction

containing the received data, signing with its private key issued by one of the authorized *Certified Authority* (CA), and forwarding it to the Hyperledger Fabric endorsement nodes (1). The Endorsement nodes execute the requested chaincode function with the received parameters, sign the result, and send it back. When the minimum number of signatures defined in endorsement policies are collected, the transaction is sent to ordering nodes (2) [Hyperledger 2022a], which will generate a block containing ordered transactions, sending it to all blockchain nodes (3). These nodes will validate the received transactions based on the blockchain policies and store the results in the *ledger* (*i.e.*, CouchDB) (4). The described process run in a Hyperledger Fabric secure communication channel, created during the blockchain implementation, to which the actor's nodes that compose the network are connected. This channel runs the chaincode containing a set of functions, allowing the interaction and execution of application-specific tasks. To demonstrate Event2ledger functionality the available chaincode functions include collected events creation and visualization. Table 2 summarizes the main configuration parameters.

Table 2. Blockchain parameters implemented in proposed scenario.

Parameter	Value
Number of channels	1
Chaincode language	Go
Ledger model	CouchDB
Endorsement policy	Major Endorsement
Transactions per block	50
Consensus mechanism	Raft

The proposed blockchain configuration includes the creation of only one channel, to which all nodes are connected. The solution uses CouchDB as the blockchain ledger, allowing the elaboration of complex queries. The endorsement policy adopted requires transactions to be validated by the majority of participants (*i.e.*, at least two, in a proof of concept scenario). The block size can have a maximum number of 50 transactions, and the consensus mechanism adopted is Raft, the Hyperledger Fabric default option. Raft is a *Crash Fault Tolerant* (CFT) mechanism operating in a permissioned network that can perform transaction ordering with low resource consumption.

4.3. Collectors

The Event2ledger collector model is build as a containerized application using a predefined script (available at the Event2ledger Github repository), ensuring the correct installation of all necessary dependencies. Due to its simplicity, this component can be adapted to interact with different data sources, converting the received data into transactions and sending to the blockchain. Each collector instance has an asymmetric key pair used in authentication and transaction signature. In the proof of concept scenario collector has two functions: (i) connect to Docker *daemon* events endpoint to receive all container life cycle events, and (ii) send collected data as well-formatted transactions to be ordered, validated and stored by blockchain nodes.

As Docker and Hyperledger Fabric blockchain supports many integration alternatives like API and via CLI, this work chose to use a CLI based collector, both for communication with Docker and with blockchain nodes. Thus, after starting the collector container, the collection of events is done through the command *docker events –filter 'type=service' –filter 'type=container' –format '{{json .}}'*, which allows the collector to receive specific events from services and containers in JSON format. The sending is

done through Hyperledger Fabric CLI command *peer chaincode invoke*, followed by the necessary arguments, among which the access path to the collector's private key, used in the transaction signature, also the endorsement node addresses and the collected event data. As a result, every service or container event generated by Docker is collected and validated, and the resulting signed transaction is stored in the blockchain for further analysis.

5. Solution Analysis

The objective of this work is to demonstrate the viability of event2ledger, implemented in a cloud-container virtualization environment, with or without a container orchestrator. This implementation uses Docker Swarm, but other orchestrating tools like Kubernetes can be used. The solution can also be implemented without an orchestrator, as it connects directly with Docker *daemon*. In proof of concept scenario, the goal is to collect events of "service" and "container" types (*i.e.*, referring to containers and applications life cycle) directly from the local Docker *daemon*. To assess the viability of Event2ledger, two test scenarios were designed:

1. **Generation of "service" events through application life cycle simulation:** Simulate an application life cycle comprising the creation, resizing, and finalization of an application. For this test, an application is created and executed as a single replica. Then, the number of replicas is changed to three, starting with two new instances. When there are three replicas in execution, the application is finalized.
2. **Container events generation through CLI:** Since the collector connects directly to the local Docker *daemon*, the event capture performed by event2ledger collectors is independent of any orchestrator solution. To validate this functionality, the container life cycle transition should be performed through Docker CLI, specifically those related to the creation, pause, resume, and finalization.

In either case, the expected result is that all generated events should be collected by event2ledger collectors running on the node where it was generated. For each event, a corresponding transaction must be generated, signed, ordered, and validated, and its results stored in the ledgers and accessible through a chaincode query. After the proposed tests execution, the demonstration presents how to interact with the blockchain and retrieve collected data and other metrics about the installed chaincode and transactions.

Code 1: Transaction containing collected event.

```
{ "Transaction ID": 7deciw95xncg3in
"Validation Code": 0
"Payload Proposal Hash":
  a52ee818c2dee32aa214154ee86a52ee81
"Endoser": {"ProviderMSP","DeveloperMSP","UserMSP"}
"Chaincode Name": eventdb
"Type": ENDORSER_TRANSACTION
"Value": {
      "Type": "service",
      "Action": "create",
      "Actor": {
            "ID": "ciw95xncg3in1pbte8h2kmi6a",
            "Attributes": {
                  "name": "appdemo"
                  }
            },
      "scope": "swarm",
      "time": 1625360377,
      "timeNano": 1625360377017468923
      "sender": "e2l_manager"
      },
"Timestamp": "2021-01-30 16:25:37.356 +0000 UTC",
"IsDelete": "false" }
```

Code 1 presents a transaction containing a collected event. This transaction contains in its payload the collected event and additional data regarding the transaction itself,

such as the validation code, the payload hash, and the timestamp among other relevant information. During the proof of concept, scenario execution was possible to identify the successful collection of all generated events as the respective validated transactions through the interaction with blockchain query functions and with the Blockchain Explorer tool, as described in detail in Event2ledger documentation, available in the repository.

6. Considerations & Repository

The proposed solution has as a differential the implementation of an append log solution using a consortium blockchain, offering high flexibility by allowing the collection of different data types from multiple data sources. The solution also offers traceability of every collected event through the collector and endorsers' signatures and secure communication by using Transport Layer Security (TLS). Distributed storage and cryptographic chaining ensure the integrity and availability of collected data. In addition, it is also possible to customize and update the installed chaincode to implement new features, since the required policies are satisfied and agreed upon by all actors.

The event2ledger source code, containing the collectors and necessary files to create and start the blockchain is available at the author's GitHub (https://github.com/marques-ma/e2l_v0.5.git). The repository also contains the solution documentation detailing the installation, usage, and chaincode functionalities. [Marques and Miers 2021] and [Marques. et al. 2021], provide additional information about this work, including related works research and theoretical details can be found.

Acknowledgments: The authors thank the support of FAPESC, and LabP2D / UDESC. This work was supported by Ripple's University Blockchain Research Initiative (UBRI) and in part by the Brazilian National Council for Scientific and Technological Development (CNPq - grant 304643/2020-3).

References

Dawadi, B., Shakya, S., and Paudyal, R. (2017). Common: The real-time container and migration monitoring as a service in the cloud. *Journal of the Institute of Engineering*, 12:51.

Docker (2022). Docker swarm services. https://docs.docker.com/engine/swarm/services/.

Hyperledger (2022a). Hyperledger fabric endorsement policies. https://hyperledger-fabric.readthedocs.io/en/release-2.2/endorsement-policies.html.

Hyperledger (2022b). Hyperledger fabric prerequisites. https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html.

Marques, M. and Miers, C. (2021). Event2ledger: Container allocation and deallocation traceability using blockchain. Master's thesis, Santa Catarina State University, UDESC.

Marques., M., Miers., C., and Simplicio Jr.., M. (2021). Container allocation and deallocation traceability using docker swarm with consortium hyperledger blockchain. In *Proceedings of the 11th International Conference on Cloud Computing and Services Science - CLOSER*, pages 288–295. INSTICC, SciTePress.

Mouat, A. (2015). Using docker: Developing and deploying software with containers.