Interface for the deployment of blockchain applications using the Katena framework

Biagio Marra, Raffaele Russo

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy

Abstract. KATENA is a framework for the deployment and management of Blockchain applications, in particular compatible with Ethereum, a widely used Blockchain technology. The framework addresses the challenges of managing Blockchain applications, which often rely on adhoc solutions that are difficult to maintain and error-prone. The paper proposes a web application interface that enables the deployment of Blockchain applications via the Katena framework, making use of Ganache for local network creation and xOpera for topology deployment.

Keywords: Blockchain · Deployment · Katena · xOpera · Ganache · Smart Contract.

1 Introduction

Blockchain has emerged as an innovative technology that marks the transition from centralised to decentralized systems, offering new standards of security, transparency and control. It acts as a distributed digital ledger, where transactions are stored on a computer network, rather than being managed by a single central entity. This creates a shared database that all participants can view and update, following established rules to validate transactions, making it more resistant to fraud and cyber attacks. As the blockchain has evolved, its applications have expanded beyond cryptocurrencies, extending to different sectors through smart contracts. These are computer programmes that automatically perform specific actions when certain conditions are met. However, although smart contracts are publicly accessible and their code cannot be modified after publication, this represents a limitation in terms of correcting any vulnerabilities. [1]

In this context, Ethereum emerges as a leading platform for the creation of decentralized applications through the use of smart contracts, which can be upgraded to overcome the limitations of traditional unmodifiable contracts. The platform is open to all, allowing developers to build applications without the need for intermediaries, thus ensuring greater transparency and autonomy. At the heart of Ethereum is the Ethereum Virtual Machine (EVM), a virtual environment that securely executes smart contracts on all nodes of the network. [2] [3]

Every transaction on the blockchain, including the deployment of smart contracts, requires the payment of a fee, known as 'gas,' which is used to compensate the miners or validators that process the transactions. This gas, paid in the network cryptocurrency (such as Ether for Ethereum), can be expensive, especially when managing complex contracts. If the deployment fails, the gas used for the transaction is not refunded, causing significant monetary losses.

Furthermore, manually managing the deployment of a series of interconnected smart contracts can be complex and error-prone. The lack of tools that automate and simplify this process is a major challenge in the development of decentralized applications.

In this scenario, KATENA presents itself as a declarative framework designed to simplify the deployment and management of decentralized applications on Ethereum and other Ethereum Virtual Machine (EVM) compatible platforms. Using a model-driven approach (Model-Driven Engineering), KATENA enables developers to create and manage complex applications by significantly reducing the code needed and automating the deployment process. This framework offers reusable components to define the structure of applications, including both on-chain (such as smart contracts) and off-chain elements. [4]

This paper focuses on the implementation of a graphical user interface developed to optimise the deployment process and the verification of smart contracts within the KATENA framework. The interface allows the loading of topology files, `JSON` files of contracts and the source code of the smart contracts, displaying the Deployment Log in real time. During this procedure, the deployment is performed using Ganache and XOpera. In addition, Blockscout, a dApp that provides a detailed dashboard to monitor transactions and deployed contracts, has been integrated; it also allows direct interaction with smart contracts.

2 Background

The following sections introduce the basic concepts needed to fully understand our solution: blockchain, smart contracts and the different tools used for the deployment process.

2.1 Blockchain

The Blockchain is a decentralized network of nodes that maintains a register of transactions in a sequence of blocks, along with a shared state resulting from the executed transactions. Each transaction can be monetary (e.g. Alice sends 10 Ethers to Bob) or require the execution of smart contracts, programmes stored and executed on the Blockchain.

A key aspect of the Blockchain is its ability to solve the 'Byzantine Generals Problem' of validating shared state in a trustless environment. Each node keeps a local copy of the Blockchain, used to validate new transactions, and a decentralized consensus mechanism, based on cryptography and incentives, determines which transactions are valid. Verified transactions are pooled into a new blockchain and honest participants are rewarded.

To carry out transactions, users must have a digital wallet with a public key (wallet address) and a private key (to sign transactions). Blockchains, like Ethereum, support smart contracts written in languages such as Solidity or Vyper. These contracts have an object-like structure and can use reusable function libraries.

When contracts are completed, their code is compiled into executable bytecode by the Ethereum Virtual Machine (EVM), and an Application Binary Interface (ABI) is generated for interaction between on-chain and off-chain components. The distribution of a smart contract is done by sending a transaction with the bytecode, and once accepted, the contract is stored in a specific address on the Blockchain.

The execution of a smart contract has a cost in native currency, which must be paid to avoid overloading the network. Only users can initiate execution by creating transactions, during which contracts can invoke other contracts. [5] [6]

2.2 Deployment of a blockchain application

Blockchain network deployment can be done in different configurations, depending on the project goals and available resources. The two main options are local deployment and public deployment.

Local deployment is ideal for developers who want to test their applications in a controlled and cost-free environment. Tools like Ganache, Hardhat, and Truffle allow you to simulate a local blockchain, making it easy to develop and test smart contracts without the complexities of public networks. Ganache, for example, provides a customizable and fast blockchain, allowing you to send transactions and monitor calls to contracts. Hardhat offers a development environment with debugging capabilities, while Truffle provides a suite of tools for migration and dependency management of smart contracts.

On the other hand, public deployment consists of deploying smart contracts on networks that are accessible to everyone, such as Ethereum or Binance Smart Chain. This choice is suitable for applications that require a high level of decentralization and accessibility. Options include mainnet networks, where smart contracts run in a real environment and require gas fees, testnet networks used to test features with no associated costs, and private networks accessible only to specific users, useful for projects that require more privacy and control.

The choice between local and public deployment depends on factors such as the purpose of the project, costs, security, and the need for testing; in general, local deployment is preferable for development and testing, while public deployment is suitable for interaction with users and the market. Once the deployment method is chosen, the next step is to write smart contracts, which are autonomous programs that run on the blockchain. Developers use specific languages such as Solidity for Ethereum or Rust for Solana. Smart contracts follow an object structure, containing state variables, functions, and constructors that define the operational logic.After being written, smart contracts must be compiled. This process transforms the source code into bytecode that can be executed by the Ethereum Virtual Machine (EVM). During compilation, a

`JSON` file is generated containing crucial elements for interacting with the contract: `ABI` (Application Binary Interface), which defines the contract's functions and events; bytecode, the compiled contract code; contract address, which is assigned to the contract once deployed; and constructor parameters, which facilitate initial deployment. Before a contract can be used, it is essential to verify it. Verification consists of validating the contract's source code against the bytecode deployed on the blockchain, thus ensuring transparency and trust in its functionality. An innovative aspect of deploying and managing blockchain applications is represented by KATENA, a framework that allows you to declare the application topology using TOSCA (Topology and Orchestration Specification for Cloud Applications). This approach facilitates the deployment and management of blockchain components, allowing developers to more efficiently model the relationships between smart contracts and infrastructure resources. Since smart contracts are immutable once deployed, it is essential to implement management strategies for any bugs or vulnerabilities, such as upgradable smart contracts, which use design patterns to allow for updates without data loss. To facilitate interaction with smart contracts and viewing the blockchain, dApps (decentralized applications) play a crucial role. DApps provide a user-friendly interface for users, allowing them to easily interact with contracts and access features such as viewing transactions, managing wallets, and sending transactions. With these applications, users can utilize the blockchain without needing to understand the underlying technical details. Additionally, many dApps incorporate verification mechanisms to ensure that smart contracts have been validated and that interactions are secure and reliable. Finally, network monitoring is essential to ensure that smart contracts are working as intended and to maintain the security and performance of the network. Using analytics and monitoring tools allows for early detection of issues, ensuring optimal operation of the blockchain network. [7] [8] [9]

3 Approch

3.1 Winery extension attempt with Katena

Winery is an OpenTOSCA tool designed for modelling and managing microservice-based applications. This tool offers an intuitive graphical interface for creating models using the TOSCA language, allowing users to define the topology and relationships between the various components. Winery facilitates integration and orchestration in cloud environments, promoting the reusability of models and simplifying the deployment of decentralised applications (dApps) and services in the cloud. [10]

In our attempt to use Winery, the goal was to integrate the nodes, relationships and capabilities present in KATENA, such as ‘contracts’ and ‘network’ nodes. This approach would have allowed the creation of a TOSCA topology using the integrated nodes. In addition, it was planned to develop a functionality for the deployment of the topology through xOpera, in order to facilitate the management and orchestration of blockchain applications.

The loading and representation of nodes in Winery was successfully completed, including the definition of requirements, properties, capabilities and dependencies between the various components. However, difficulties arose during the integration of the deployment functionality. Specifically, the way Python scripts were called from KATENA nodes was not compatible with the way Winery required nodes to be created and managed. This mismatch made the implementation of the topology deployment process complex.

3.2 BlockVerse deployment

The functionality of deploying a blockchain in the application is articulated in several key elements.

- **UI interface** implemented in the BlockVerse application, allows users to upload the necessary files and monitor the status of the deployment process in real time, also showing links to connect to the BlockScout dApp.
- **Server Python** receives `POST` requests from the web interface, processes them and sends the appropriate commands to be executed in the container hosting the Katena framework.
- **Katena Docker** where all the tools needed to correctly deploy the blockchain topology are present.
- **Blockscout** dApp that allows users to view all information related to the implemented blockchain and interact with verified contracts, providing a complete management and control experience.

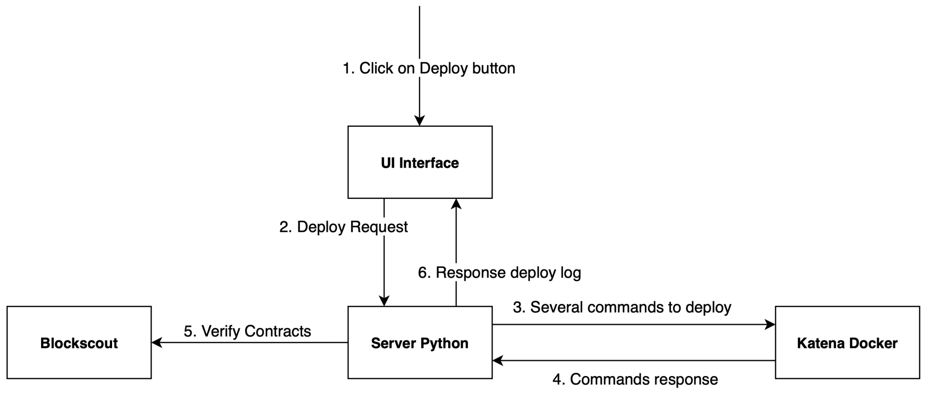


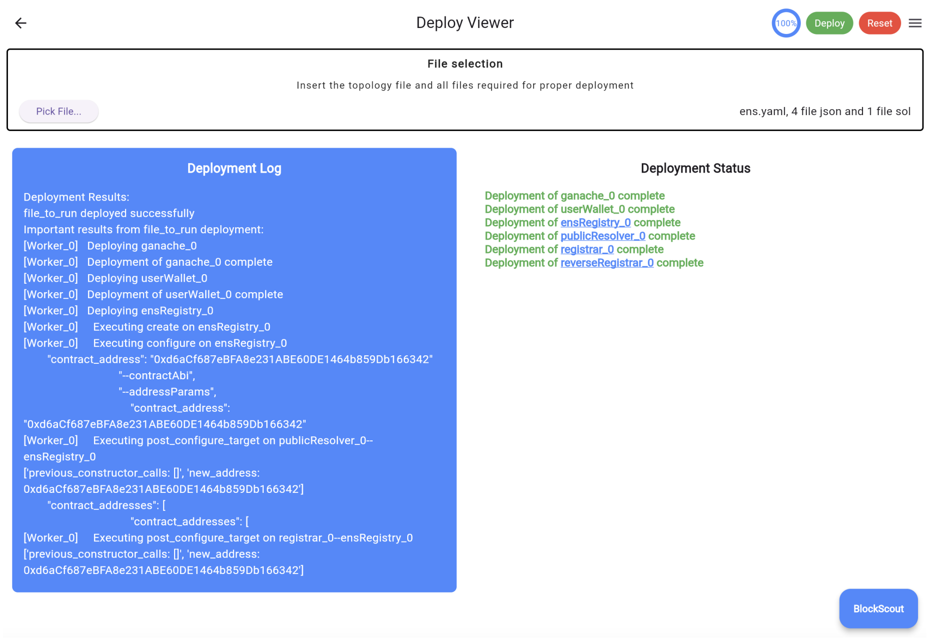
Fig. 1. High-level component interactions

3.2.1 UI Interface

The UI Interface, developed using Flutter and Dart, allows the upload of local files necessary for the deployment of the blockchain. On the main screen, there are buttons for uploading `YAML` files, containing the blockchain topology, `JSON` files, representing the ABIs of the contracts included in the topology, and Solidity files, necessary for verifying the contracts once the deployment is complete.

Once the deployment process has started, the interface sends an HTTP request `POST` to the server to perform the deployment and establishes a connection via WebSocket on a specific port to monitor the status of the various completed contracts.

The interface also includes a panel displaying the deployment logs upon completion, and another panel displaying the real-time deployment of the various nodes, providing direct links to the contracts in the BlockScout dApp, allowing interaction with them. Lastly, there is a circular indicator to monitor the status of the process, a `Deploy` button to start the deployment (with the possibility of running multiple deployments on the same network), a `Reset` button to reset the network and delete the entire loaded blockchain, and a BlockScout button that directs to the application's home page.


Deploy Viewer
100%
Deploy
Reset
File selection
Insert the topology file and all files required for proper deployment
Pick File...
ens.yaml, 4 file json and 1 file sol
Deployment Log
Deployment Status
BlockScout

```
Deployment Results:
file_to_run deployed successfully
Important results from file_to_run deployment:
[Worker_0]   Deploying ganache_0
[Worker_0]   Deployment of ganache_0 complete
[Worker_0]   Deploying userWallet_0
[Worker_0]   Deployment of userWallet_0 complete
[Worker_0]   Deploying ensRegistry_0
[Worker_0]     Executing create on ensRegistry_0
[Worker_0]     Executing configure on ensRegistry_0
       "contract_address": "0xd6aCf687eBFA8e231ABE60DE1464b859Db166342"
                        "--contractAbi",
                        "--addressParams",
                           "contract_address":
"0xd6aCf687eBFA8e231ABE60DE1464b859Db166342"
[Worker_0]     Executing post_configure_target on publicResolver_0--
ensRegistry_0
['previous_constructor_calls: []', 'new_address:
0xd6aCf687eBFA8e231ABE60DE1464b859Db166342']
       "contract_addresses": [
                           "contract_addresses": [
[Worker_0]     Executing post_configure_target on registrar_0--ensRegistry_0
['previous_constructor_calls: []', 'new_address:
0xd6aCf687eBFA8e231ABE60DE1464b859Db166342']
```

```
Deployment of ganache_0 complete
Deployment of userWallet_0 complete
Deployment of ensRegistry_0 complete
Deployment of publicResolver_0 complete
Deployment of registrar_0 complete
Deployment of reverseRegistrar_0 complete
```

Fig. 2. Dashboard for Deploy

3.2.2 Server Python

The Python server handles logistics and calls to the Docker container, exposing two URLs on port `5001` and a websocket on port `8765`.

The first URL, `/deployment`, allows you to deploy the blockchain through a `POST` request that accepts a YAML file, `JSON` file, and `Solidity` file. When this call starts, the server sends a command to clean the deploy.log file in the container, followed by loading the `YAML` file and all `JSON` files. The `YAML` file is loaded with a direct command, while loading the `JSON` files requires additional operations due to the byte limit imposed by Docker.

Then, the server checks whether a Ganache network already exists in the container. If so, it sends the command to deploy the topology on it; otherwise, it sets up a new Ganache network and proceeds with the deployment of the topology. During this phase, the Python server monitors the `deploy.log` file using the `tail` command until the deployment process is complete, communicating in real time via WebSocket when a new contract has been successfully deployed.

Once the deployment is complete, the server reads the final filtered `log`, collects the addresses of the various contracts, and sends a contract verification request to BlockScout via the `API`. Finally, it returns the final filtered `log` to the user.

The second `URL`, `/reset`, sends a command to run a script that completely deletes the network in the docker container and the Blockscout app.

3.2.3 Container Katena

For the Katena Container, a modification process has been undertaken starting from the original project available on GitHub. [11]
The main changes made include the addition of a configuration file in which the topology to be executed each time is overridden, the creation of a dedicated folder to upload all the necessary `JSON` files, and the introduction of a `.txt` filter file for filtering the `deploy.log`, thus making it easier to analyze the deployment logs.
A `.sh` file has been implemented that allows to perform a network reset through specific commands, such as removing the `.opera` folder and exiting the `ganache-cli` process.
In the Ganache build function in the `bootstrap-dev.sh` file, the `-host 0.0.0.0` option has been added, ensuring visibility for BlockScout.
Finally, in the `/deploy-bench.sh` file, significant improvements have been made, including moving the `JSON` files from the `tempABI` folder to `contracts`, adding commands to filter the `deploy.log`, with the results saved to the `.txt` file mentioned above, and cleaning up the various folders used during the deployment process.

3.2.4 Blockscout

Blockscout is an open-source, containerized blockchain explorer designed to interact with Ganache-based networks. Its main function is to acquire all the information from the nodes on the Ganache network through port `8545` and present it through its own intuitive user interface. Additionally, when the Python server sends a verification request, BlockScout handles that request by performing a thorough check. This process involves comparing the bytecode contained in the `JSON` file with the source code written in `Solidity`. Once the verification is complete, users can interact with the contract, having the ability to call its functions directly from the BlockScout interface. [12]

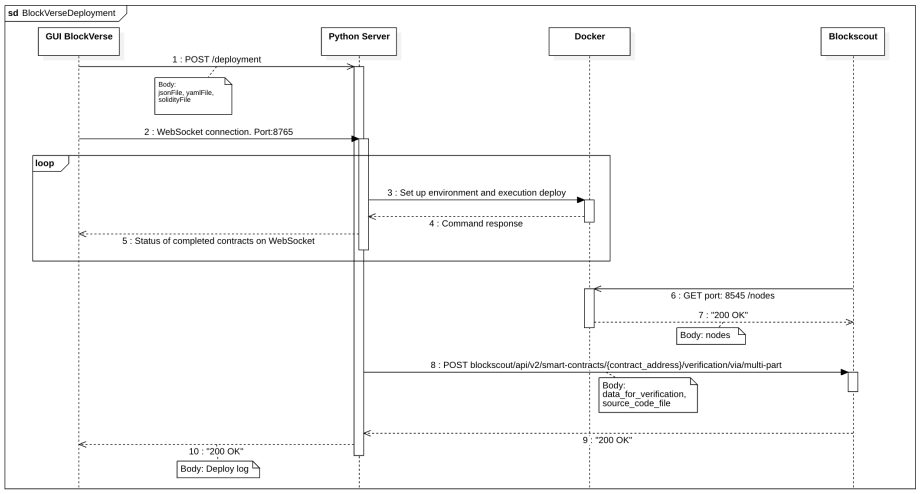

sd BlockVerseDeployment
GUI BlockVerse
Python Server
Docker
Blockscout
1 : POST /deployment
Body: jsonFile, yamlFile, solidityFile
2 : WebSocket connection. Port:8765
loop
3 : Set up environment and execution deploy
4 : Command response
5 : Status of completed contracts on WebSocket
6 : GET port: 8545 /nodes
7 : "200 OK"
Body: nodes
8 : POST blockscout/api/v2/smart-contracts/{contract_address}/verification/via/multi-part
Body: data_for_verification, source_code_file
9 : "200 OK"
10 : "200 OK"
Body: Deploy log

Fig. 3. Interazione fra i vari componenti

4 Conclusions

In conclusion, the project has successfully integrated key components for deploying and managing a blockchain, using technologies such as Ganache, xOpera, Katena and BlockScout. Thanks to a web interface developed in Flutter, users can upload `YAML`, `JSON` and `Solidity` files to start the deployment through the Python server, which interacts with the Katena located inside the Docker container. The system is able to monitor the deployment process in real time and verify contracts through BlockScout. This implementation has improved interactivity with Smart Contracts. Future projects include the goal of extending the blockchain deployment not only on the private Ganache network, but also on public networks, thus expanding the possibilities of interaction and use of decentralized applications.

References

1. Osservatori Digital Innovation. Blockchain: spiegazione, significato e applicazioni. https://blog.osservatori.net/it_it/blockchain-spiegazione-significato-applicazioni, 2024.
2. Ethereum. Ethereum documentation. https://ethereum.org/en/developers/docs/evm/.
3. Osservatori Digital Innovation. Smart contract nella blockchain. https://blog.osservatori.net/it_it/smart-contract-in-blockchain, 2024.
4. Damian Andrew Tamburri Luca Terracciano Luciano Baresi, Giovanni Quattrocchi. A survey on ethereum smart contracts. https://arxiv.org/abs/2209.05092.
5. IBM. Blockchain - ibm. https://www.ibm.com/it-it/topics/blockchain.
6. IBM. Smart contracts - ibm. https://www.ibm.com/it-it/topics/smart-contracts.
7. Truffle Suite. Ganache - truffle suite. https://archive.trufflesuite.com/docs/ganache/.
8. Truffle Suite. Truffle - truffle suite. https://archive.trufflesuite.com/docs/truffle/.
9. Shardeum. Hardhat - shardeum blog. https://shardeum.org/blog/hardhat/.
10. Winery documentation. https://winery.readthedocs.io/en/latest/.
11. DEIB Politecnico di Milano. Katena - github repository. https://github.com/deib-polimi/katena.
12. Blockscout documentation. https://docs.blockscout.com.