

BlockVerse

Author: Emanuele Andalaro

Email: emanuele.andalaro@mail.polimi.it

Abstract—KATENA is a framework for the deployment and management of Blockchain applications. In particular, it focuses on applications that are compatible with Ethereum, a popular general-purpose Blockchain technology. The aim of this paper is to explain the dashboard functionalities added to KATENA, specifically focusing on topology printing, creation, and editing.

KEYWORDS

software engineering, blockchain, TOSCA, deployment, topology management.

I. INTRODUCTION

In the rapidly evolving landscape of cloud computing and decentralized applications (dApps), the need for robust frameworks that streamline development and deployment processes has become increasingly critical. In the state of the art there are some implementation for automatizing the development and deployment but they are not always simple and maintainable or they are implemented using script bash that are not always simple to understand and executable on every platform. The problem is more evident in blockchain applications where the automatized deployment is not always simple and also not efficient manageable. On the other hand, Katena introduces metamodels specifically suitable for the development and deployment of decentralized applications. By defining reusable on- and off-chain components, Katena simplifies the creation of dApps, allowing for more intuitive modeling and management of component lifecycles. Moreover, it's not always simple for the developer to use TOSCA grammar which has some challenges in the understanding and utilization for the deployment. Here BlockVerse comes. BlockVerse is a dashboard useful for exploiting all the different functionalities provided by katena considering it as an asset for the application. So, the user will be able to manage, printing and exporting all the topology it wanted to only with a click so that it will be able to create it's own topologies using the ones already present in katena or also to print them it will be able to understand what each piece of TOSCA grammar it is implementing and then after the creation it will be able to export the topology files necessary for the deployment of it's own application.

A. Goals

- Provide a dashboard for the given deployment and management framework
- Possibility of importing topologies;

- Possibility of editing a topology

B. Domain Assumption

- [D1] KATENA is an asset for the platform;
- [D2] KATENA deployment is invoked inside the application;
- [D3] The user can see the topology as graph;
- [D4] The user can export the YAML file of the topology

II. SYSTEM ARCHITECTURE AND DESIGN


For what concerns the architecture, the backend is hosted on Firebase, the authentication part is ensured by both checking the user inputs on client side and for what concerns the server side APIs provided by Firebase Authentication are used. User Data are stored inside Firebase Firestore this permits a graceful management of Data because you can exploit the features of NO-SQL databases. Here you can find an overview of the Distributed architecture.

A. the Singleton pattern

Singletons are classes which can be instantiated once, and can be accessed globally. This single instance can be shared throughout our application, which makes Singletons great for managing global state in an application.

III. GUI AND IMPLEMENTATION

Here in this section, is presented the graphic user interface of the application accordingly to the goals there are three Screens which are most important not all the screens and functionalities are presented for simplicity. This is the flow as intended by the developer. The User starts by creating an account in the Platform. Then is able to authenticate itself by inserting email and password. Then it enters in the dashboard [2](#) here he can see a chart and then also three buttons this as you can see are able to lead you on different screen inside the dashboard the Deploy Screen, Topology printing or management here in [3](#) is presented the view for management and topology.



Welcome to BlockVerse

Login

[Forgot password?](#)

[If you do not have an account, go to sign up](#)

Figure 1. User login screen

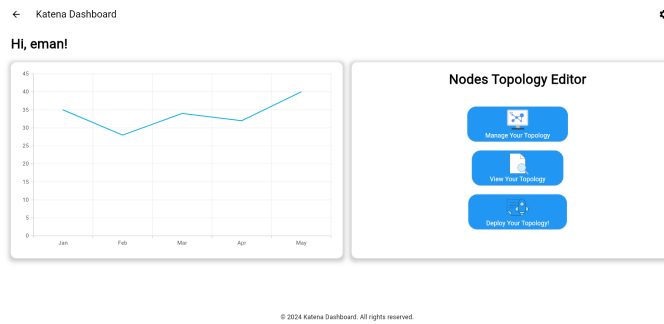


Figure 2. Dashboard screen

As an example, here is presented the snippet of a TOSCA grammar:

```

1
2
3
4
5  toska_definitions_version:
   toska_simple_yaml_1_3
6
7  imports:
8  - nodes/contract.yaml
9  - nodes/network.yaml
10
11 topology_template:
12   node_templates:
13     ganache:
14       type: katena.nodes.network.ganache
15     userWallet:
16       type: katena.nodes.wallet
17       requirements:
18       - host: ganache
19       properties:
20       privateKey: { get_input:

```

```

21       UserKeyGanache }
22       owner: { get_input: UserWallet}
23   facet1:
24     type: katena.nodes.diamond.facet
25     requirements:
26     - host: ganache
27     - wallet: userWallet
28     properties:
29     abi: "Callee"
30   diamondCut:
31     type: katena.nodes.contract
32     requirements:
33     - host: ganache
34     - wallet: userWallet
35     properties:
36     abi: "DiamondCutFacet"
37   diamondLoupe:
38     type: katena.nodes.contract
39     requirements:
40     - host: ganache
41     - wallet: userWallet
42     properties:
43     abi: "DiamondLoupeFacet"
44   myDiamond:
45     type: katena.nodes.diamond
46     requirements:
47     - host: ganache
48     - wallet: userWallet
49     - cut: diamondCut
50     - facet: diamondLoupe
51     - facet: facet1
52     properties:
53     abi: "Diamond"
54
55 inputs:
56   UserKeyGanache:
57     type: string
58     required: true
59   UserWallet:
60     type: string
61     required: true

```

Listing 1. TOSCA diamond topology

There was presented the snippet of how the diamond topology is implemented, in the figure is presented how graphically it is printed this is not only a simpler representation but a more understandable one of how the diamond topology can be represented.

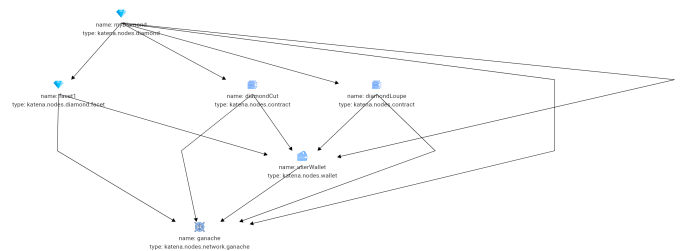


Figure 3. Topology management screen

A. Pseudo Code of the implementation

Here, are presented the three fundamental snippet which are the core of the existing application important to achieve the given goals. This is a simple pseudo-code explanation of the work done during the implementation phase. Obviously, for some functions they are declared but the pseudo code is not explained because they are intended as helper method for explaining what is happening.

[M1] Topology printing

```
1
2 function GraphPrinter()
3 {
4     Graph graph;
5     for each node in node_types
6         if check_imports(node);
7         {
8             graph.addNode(node);
9         }
10
11     for each node in graph.nodes
12     {
13         check_requirements()
14         for each requirement
15         {
16             target_node=find(
17                 targetNode);
18             add_edge(node,
19                 targetNode);
20         }
21     }
22     return graph;
```

Listing 2. Topology printer

This method prints the Topology by the means of a GraphView Library. Firstly, the TOSCA is parsed and loaded into the dashboard then the TOSCA grammar is used in order to adapt the graph components to the TOSCA grammar which is a non trivial issue but by ease of abstraction the nodes of the graph becomes Katena nodes and the requirements of each node becomes arcs of the graph.

[M2] Topology management

```
1 function GraphManagement()
2 {
3     Graph graph;
4     for each node in node_types
5         if check_imports(node);
6         {
7             graph.addNode(node);
8         }
9
10    for each node in graph.nodes
11    {
12        check_requirements();
13        for each requirement
14        {
15            capability=match_capability
16                (node, destNode);
```

```
16         add_edge(node, targetNode);
17     }
18 }
19 return graph;
20 }
```

Listing 3. Topology Management

This method is necessary for creating a new Topology or for modifying an existing one basically as the method describe before you map katena.nodes as graph nodes then this nodes in the TOSCA grammar have some requirements this, if the nodes in the requirement have the same capability or the same capability and the same type the two nodes are connectable. Some examples of connections will be provided later.

[M3] Topology exporting

```
1
2 function topologyExporting(
3     Graph graph)
4 {
5     YamlMap yamlMap;
6     for each node in graph.
7         nodes
8         {
9             yamlmap.add(
10                 node.ID);
11         }
12     for each edge in graph.
13         edge
14         {
15             if (
16                 filter_requirements
17                     (edge) &&
18                 filter_capabilities
19                     (edge))
20             {
21                 yamlMap.add(
22                     edge.ID);
23             }
24         }
25     return yamlMap;
26 }
```

Listing 4. Topology exporting

This method after the Creation of a topology exports the TOSCA of the concocted topology graph. So, as the name of the Topology suggest a graph to YAML parsing.

IV. TESTING

The testing methodologies that were used for the development of the application are several combined basically:

- Thread strategy: A thread is a portion of several modules that, together, provide a user-visible program feature;
- Bottom-up strategy: It may create several working subsystems;

A. Features Identification

The features to be implemented are described starting from the requirements. Some requirements need the implementation of new components while others doesn't require big changes

- [F1] **Sign-up and login** This feature permits to the user to sign-up and login securely using Firebase
- [F2] **Topology printing** This feature allows the user to print a topology on the screen inserting a TOSCA file on the Dashboard.
- [F3] **Topology management** This feature allows the user either to import a Topology and print it but also to modify it and also to create it's own topology.
- [F4] **Topology exporting** After the Topology creation the user can also export it's own topology by downloading a TOSCA file in his computer.

B. Component Integration and Testing

Here is explained how each component was integrated in each part of the development and how the different components were implemented and tested.

- [F1] **Sign-up and login** The only thing to be tested here is the interaction of the dashboard with the Authentication API provided by Firebase which is considered reliable by definition.

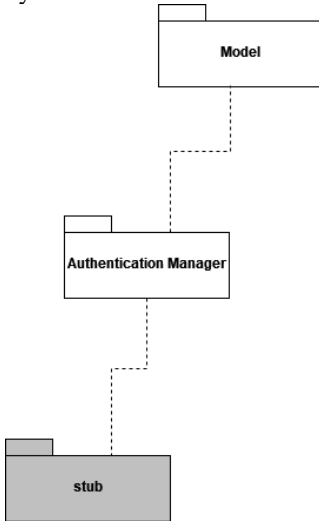


Figure 4. sign-up and login testing

- [F2] **Topology printing** Here the new part to be tested is the topology manager which is the component in charge to manage everything regarding the Topology from the printing up to the changing and the exporting.

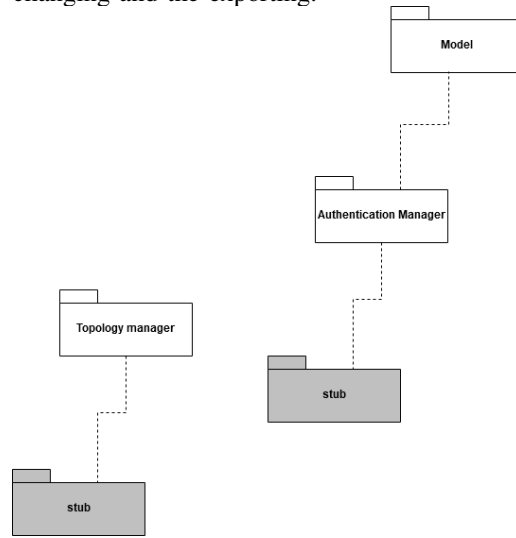


Figure 5. topology printing testing

V. FUTURE WORK

Having Dashboard for the creation and printing of topology is the first move for passing from a complex representation of the TOSCA to a simpler graphical one. But there are more difficult challenges that could be envisaged in fact in Blockchain application it's becoming a crucial issue to know where and when a node could fail a further improvement of the work done up to now could be to use generative AI to implement a sort of topology assessment which could be useful to understand when a failure could arrive. So, the idea is to use a large dataset of topology to train a model for having an assessment provided the TOSCA of the topology then the user could know when and where a node could fail based on some parameters as an example it could node positioning inside the topology or type of network utilized or moreover the use will be able to know a priori which is the percentage of the deployment failure and success based on the topology and the TOSCA provided.

REFERENCES

- [1] A Declarative Modelling Framework for the Deployment and Management of Blockchain Applications; url:<https://arxiv.org/abs/2209.05092>