2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 978-1-6654-9538-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICBC54727.2022.9805493

Decentralized Application Infrastructures as Smart Contract Codes

Rabimba Karanjai*, Keshav Kasichainula*, Nour Diallo*, Mudabbir Kaleem*, Lei Xu†, Lin Chen‡, Weidong Shi*
*University Of Houston, TX, USA
{rkaranjai, kkasichainula, ndiallo, mkaleem, wshi3}@uh.edu
†Kent State University, OH, USA , ‡ Texas Tech University, TX, USA
xuleimath@gmail.com, lin.chen@ttu.edu

***Abstract*—With the recent advance in concepts like decentralized "cloud" and blockchain-enabled decentralized computing environments, the legacy modeling and orchestration tools developed to support centrally managed cloud-based ICT infrastructures are challenged by such a new paradigm built on top of decentralization. On the other hand, decentralized "cloud" and computing infrastructures need to support many Dapp use cases. As the complexity of these targeted application scenarios increases, there is an urgent need for developing automation and modeling tools for deploying and managing decentralized infrastructures. Instead of creating such tools from scratch, a natural approach is extending mature infrastructure modeling tools for Dapps and decentralized computing environments. To this end, in this work, we have developed extensions to the TOSCA domain-specific language to support smart contract specification of decentralized computing infrastructures for supporting Dapps, where smart contracts or chain codes manage a decentralized computing environment. The result is blockchain-based orchestration and automation for decentralized "cloud" and computing environments, which is a step forward for achieving full decentralization in general-purpose computing.**

***Index Terms*—TOSCA, Smart Contracts, Blockchain, Infrastructures**

I. Introduction

Modern ICT (information and communication technology) infrastructures are becoming increasingly sophisticated and infeasible to manage manually. Various technologies have been developed to support efficient and flexible physical ICT infrastructure management, such as computer virtualization, software-defined network, and network function virtualization. To further automate the deployment, configuration, and management of centralized infrastructures like a cloud-based ICT center, system administrators, have been using orchestration tools [1] to dynamically deploy services. These orchestration tools are usually centrally managed by so-called orchestrators. An administrator fully trusts the orchestrator to manage and coordinate the lifecycles of ICT components (e.g., computation, storage, and communication resources), which constitute a deployed service. To further improve the management of the ICT infrastructure, the concept of "infrastructure-as-code" was developed [2], where a system administrator describes the target infrastructure and orchestration objectives as documents using a domain-specific language. This concept brings several advantages, such as faster configuration and environment provisioning, greater consistency, and minimized risk/enhanced security.

One of the most popular domain-specific languages for infrastructure definition is TOSCA (Topology and Orchestration Specification for Cloud Applications [3])), which helps manage and automate the cloud-based deployment of services. With TOSCA, administrators can focus on the intended functionalities of the infrastructure and only need to describe the end goal of the envisioned service deployment environment. The interpretation and implementation of the description are delegated to the orchestrator. One of the TOSCA's fundamental properties is cloud platform agnostic, which makes it ideal for managing multi-cloud ICT infrastructure. There are numerous implementations of TOSCA based on its standards (e.g., Alien4Cloud(https://alien4cloud.github.io/), Cloudify(https://github.com/c loudify-cosmo), and OpenTOSCA [4]).

With the introduction of blockchains [5], smart contracts [6] and the recent advance in decentralized "cloud" and blockchain-enabled decentralized computing environments, the legacy modeling, and orchestration tools are challenged by this new emerging decentralized computing paradigm, where computing infrastructures consist of fully decentralized components. The creation and maintenance of such infrastructures are often facilitated by the principles of crypto-finance or crypto-economy, which provides various incentive mechanisms for sustaining a decentralized ICT infrastructure. The objectives of decentralization are to eliminate centrally managed components and reduce reliance on trusted entities. These are not aligned with the original design goals of orchestration tools like TOSCA. On the other hand, decentralized "cloud" and computing infrastructures need to support a wide range of Dapp use cases. As the complexity of these targeted application scenarios increases, there is an urgent need for developing automation and modeling tools for specifying, deploying, and managing decentralized infrastructures.

To overcome these limitations and keep all the benefits of infrastructure as code, we propose Decentralized Application Infrastructure(DAI), which facilitates the automated management of decentralized computing environments and Dapps (decentralized applications) using blockchain technology. DAI also relies on a specific language to describe ways to manage the decentralized resources, which are encoded and processed in the form of smart contracts. Instead of creating such tools from scratch, DAI extends existing mature infrastructure modeling tools like TOSCA for Dapps and decentralized computing environments. Another benefit of extending TOSCA is that it potentially can support mixed ICT infrastructures that comprise both clouds and decentralized computing resources. The efforts bring a missing piece of the puzzle for realizing fully decentralized computing infrastructures.

In summary, the contributions of this work include: **(i)** We have developed a detailed design of DAI, which enables provisioning and management of decentralized ICT infrastructures with a unified, general-purpose, comprehensive and standard-based infrastructure modeling tool; **(ii)** The new tool supports modeling, orchestration, and management of decentralized computing environments by leveraging blockchain technology and smart contracts. **(iii)** We have provided examples to demonstrate the usefulness of DAI; and **(iv)** We have implemented a prototype of DAI and conducted preliminary experiments on it to demonstrate its practicality.

II. Background

Decentralized "cloud" The concept behind decentralized "cloud" is to enable a new paradigm of computing infrastructures based on decentralization and blockchain facilitated economic models. Different from the traditional cloud computing model that depends on very few trusted service providers, decentralized "cloud" leverages peer-to-peer based nodes for computing (e.g., [7], [8], [9], [10], [11], [12]. These systems often apply blockchain or distributed ledger-based designs for providing incentives to the users who contribute resources, for instance, using utility tokens as payment for computing resources. The proposed platforms can target either general-purpose computing, for instance, Dfinity [12] , or specific applications like databases, artificial intelligence, or machine learning-oriented use cases (e.g., [13]). To address privacy concerns, some propose to use trusted execution environments (TEEs) [14] for processing data.

Infrastructure modeling and TOSCA For deploying, configuring, and managing services and software in cloud-based environments, system administrators apply automation tools for provisioning server nodes and instantiating software on them. Typically, this is achieved by using domain-specific language to describe the targeted infrastructures as templates and their orchestration intents in policy files. There are several such languages backed by the industry and cloud service providers [15], [16]. TOSCA is one of the standard-based modeling languages for handling cloud-based infrastructure deployments and orchestration [17]. It was developed by OASIS (Organization for the Advancement of Structured Information Standards) as a cross-cloud standard. Many cloud modeling languages are compatible with TOSCA.

Smart contracts A smart contract, also called a "chain code" in the context of Hyperleger Fabric, is software that can function as a general-purpose application to interact with blockchain data. Smart contracts implement the business logic of a blockchain application. Smart contracts are executed by blockchain nodes and verified using consensus. Many smart contract languages have been developed in the past few years. In the case of Fabric, it provides an open-source SDK that supports chain code written using popular programming languages such as GoLang, Nodejs, and Java.

III. Overview of DAI

The following section will cover the details of each component and demonstrate the model with examples. Specifically, the requirements of our efforts include:

- *Decentralized deployment and management of blockchain-based ICT infrastructures for supporting Dapps.*
- *Protection of infrastructure descriptions, configurations, settings, policies, and meta-data with the security guarantees of blockchain-based system.*

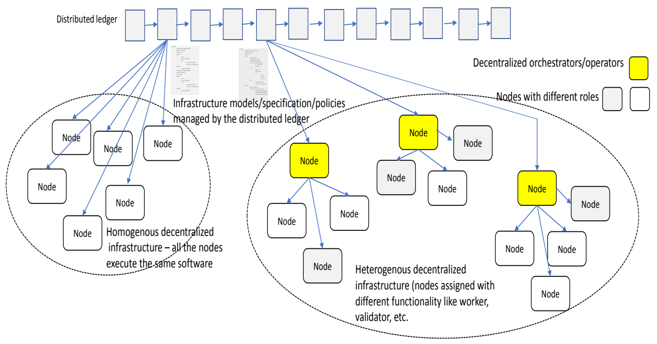

Distributed ledger
Decentralized orchestrators/operators
Nodes with different roles
Infrastructure models/specification/policies managed by the distributed ledger
Node
Homogenous decentralized infrastructure – all the nodes execute the same software
Heterogenous decentralized infrastructure (nodes assigned with different functionality like worker, validator, etc.

Figure 1: Overview of DAI.

- *Compatibility with the existing orchestration standards such as TOSCA.*
- *Universal and generic modeling support for various applications and use cases, which means that the tools will not be restricted to a specific Dapp or blockchain system.*

There are challenges to developing a framework that can meet these requirements.

Since they are not designed for blockchain-based applications and decentralized environments. They often consider cloud-based economic models instead of crypto-economic concepts and principles.

Figure 1 provides a high-level overview of the environment. A blockchain/distributed ledger is utilized for managing components and services required by a Dapp. It is based on extending declarative-oriented languages like TOSCA and implementation of the extended language as smart contracts or chain codes. Governance entities of Dapps focus on specifying the intended infrastructure or envisioned operating environment as a set of on-chain data objects described using smart contracts/chain codes. Participating nodes of the Dapps will retrieve descriptions from the blockchain, assemble the data objects into deployment documents (e.g., TOSCA instance templates), deploy and configure the local node/nodes by interpreting and realizing the extended TOSCA documents. Instead of having centrally managed orchestrators, participants are independent and autonomous in making decisions (viewed as incentive-driven agents).

All the smart contract data related to a decentralized infrastructure can be updated following governance principles. Virtual machine and container images can be stored off-chain, for instance, using IPFS [18]. Links to the off-chain images are stored on-chain. On-chain data objects related to a decentralized infrastructure can be stored as key-value pairs that can be realized using maps or dictionaries. To import and export infrastructure documents, the system supports either YAML based or JSON based documents. For a smart contract, an import function parses the received YAML document and stores the retrieved key-value pairs into on-chain data objects. An export function converts on-chain data objects into a YAML or JSON document.

Figure 2 illustrates the flow of the decentralized infrastructure deployment process. Concepts of the TOSCA modeling standard are extended to adopt both abstract views of a decentralized infrastructure topology (type-level TOSCA) and the implementation level of nodes (instance level TOSCA). For a Dapp environment, type-level documents encapsulate requirements, policies, governance rules and offer a high-level view of the decentralized infrastructure topology.

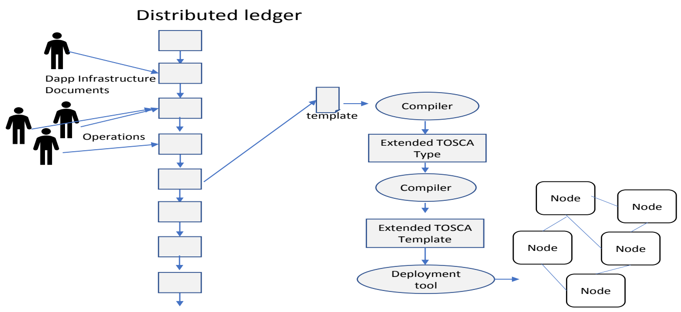


Figure 2: Workflow of orchestrating decentralized infrastructure.

For a Dapp project, a governance entity can create initial documents reflecting an abstract view of the Dapp infrastructure. Then, the records are stored as on-chain data using multiple smart contracts (detailed explanation with examples in Section IV). The governance entities can update the documents using blockchain transactions. Participants can retrieve relevant data from a Dapp environment from the blockchain and assemble the data into comprehensive TOSCA documents.

Using TOSCA compiler, the documents will be translated to instance level model that can be used for implementing nodes in a decentralized system. Infrastructure documents stored on-chain can be modified over time. Local nodes synchronize with the blockchain. Infrastructure updates can trigger a reconfiguration of local nodes.

The proposed framework ensures that Dapp administrators deploy change in the form of infrastructure change/reconfiguration. It will follow the process below:

- *The changes are written into a TOSCA based transaction. The extensions proposed in this paper works as a placeholder for the rules set for the deployment.*
- *The proposed changes will be vetted through on-chain governance and/or permissions (e.g., voting for adoption of the changes, verification of signatures, checking policies).*
- *A parser takes these changes and writes them into smart contracts executed into the change.*
- *Once validated using consensus, on-chain states will be updated with the new infrastructure specification, protected by the blockchain.*
- *A network of operators connecting to the blockchain will export the states, assemble them into TOSCA documents and convert them into actionable building blocks by a TOSCA interpreter.*
- *These are then deployed as changes into the targeted system.*

The above steps ensure that the infrastructure changes are not bound to any centralized component prone to failure and provide audit-ability with integrity protection.

IV. Detailed Design of DAI

We present the detailed design of major components of DAI in this section.

A. Assumptions

In this work, we make certain assumptions about the decentralized computing environment. **Firstly**, we assume a blockchain or distributed ledger with smart contract support applied for managing Dapps and decentralized computing infrastructures. The blockchain records a sequence of transactions in append-only mode. As a result, it comprises ordered blocks created from a genesis block under a consensus mechanism (e.g., PoW, PoS [19], PoA, PBFT [20], DPoS [21]). Each block has a block height. Further, we assume that blocks are generated with a relatively constant speed. Details of consensus mechanism and block structure are irrelevant. In this work, we use Hyperledger Fabric as a target blockchain environment. However, the concept and framework can be extended to other blockchains. We also assume that the blockchain supports smart contracts. It is plausible to adapt the model to blockchains without smart contracts where data objects are stored off-chain, and only hash values are kept on-chain. Such work is outside the scope of this paper. **Secondly**, we assume that multiple Dapps can share the identical blockchain for infrastructure orchestration in this environment. How this orchestration blockchain manages its own participating nodes or infrastructure is outside the focus of this paper. **Thirdly**, we assume that the underlying blockchain system for supporting consensus and network communications is secure and reliable, which means that measures have been taken to prevent attacks at separate layers. Such attacks include but are not limited to, for instance, 51% attack, side-channel attacks, Sybil attacks, attacks to DNS, hard forks, quantum attack, attacks due to smart contract language design flaws, etc. **Fourthly**, we assume that the blockchain is used for transactions of tokens and assets where users may own portfolios of such assets. Thus, the participants of the Dapps and the decentralized infrastructures are only motivated by financial incentives.

Table I: Overview of extension to TOSCA core concepts.

TOSCA features	**Changes**	**Smart contract modules**
Metadata	Introduction of new data attributes related to decentralized environment and crypto-economic based computing models like stake.	core, mapper
Node_type	New node types like operators, validators, proxy, and new characteristics like TEE support, cloud vs. non-cloud nodes.	identity, topology, node_templates
Node_template	New node templates for Dapps and blockchain enabled infrastructures like worker nodes, validator nodes, proxy nodes, etc.	node_templates
Capabilities	New capabilities are supported like TEE capability.	topology, node_templates
Relationship	New relationships are defined like composed_of, operated_by.	topology, node_templates
Policies	New policies are added like approaches for managing decentralization, usage of stake in infrastructure management.	topology, mapper

B. Extensions to TOSCA Concepts

In our system, governance entities describe their infrastructure requirements, envision topology, security settings, and orchestration intents as template files, using extended TOSCA language. These files are imported to the blockchain via transactions with data integrity and authenticity protected by the public keys of the transaction submitter. After verification, the submitted data is stored as on-chain states.

The current TOSCA specification is designed to support cloud-based environments with centrally managed orchestrators. Most of the key TOSCA concepts need to be extended to support crypto-economy-based and decentralized topology. A summary of these changes is listed in Table I.

Table II: Smart contract modules.

Modules	Features
Governance	Implementation of governance logic like voting for changes.
Core	Essential and common settings for Dapps and decentralized computing infrastructures including links to other contract modules.
Names & identifiers	Credentials (e.g., X509 certificates), user accounts (wallet addresses) including operator lists, identifiers.
Artifacts	TOSCA artifacts including links to virtual machine and container images.
Decentralized infra-structure specification & topology	Topology and type data of decentralized infrastructures.
Infrastructure templates	Template data of decentralized infrastructures after mapping.
Mapper	Smart contract that allocates operators to abstract infrastructure specifications.
Security	Security policies and configuration settings like blacklists, permissions.

Although the extensions of TOSCA involve the introduction of new data attributes, node types, and policies like stakes, operators, decentralization-based optimization, the described changes to the domain-specific language itself are non-intrusive. They can be applied within the scope of the original language. This means that users do not need to modify the core template framework of TOSCA. The extended TOSCA with decentralization and Dapp support maintains the actual structures of TOSCA documents and are backward compatible with the TOSCA core concepts like types, instances, policies, requirements, and capabilities.

C. Smart Contract Modules

Instead of using a single, smart contract or chain code program, the management of decentralized infrastructures is split into multiple smart contracts. Each component can further be recursively decomposed into multiple contract codes. Each smart contract program only maintains a subset of an infrastructure model. This practice is common in smart contract design with certain advantages, such as reduced footprint of the individual contract, the lower risk caused by compromised contracts, easier upgrade, etc. The set of smart contracts realizes a single namespace for a decentralized infrastructure.

There are several ways that on-chain states can be updated: (i) voting by the governance smart contract; (ii) update by an authorized user, for instance, original creator of the smart contract; (iii) update by a set of authorized users with a scheme like multi-signature.

D. Use Cases of DAI

We will use two use cases to demonstrate the way the extensions work.

1) Sharding Network: We consider a simple sharding-based peer-to-peer computing network. The nodes are called workers. To prevent certain attacks, a stake is required for joining the network. Additional security enhancement can include measuring reputation scores or negative incentives like deposits. The network is partitioned into multiple shards where worker nodes are randomly allocated to the shards. Time is divided into epochs, and worker nodes are re-shuffled between epochs. The structure is abstract enough to capture many use cases like using the worker nodes for out-sourced computation or smart contract execution (e.g., [22]).

Governance The system includes a group of smart contracts that exist on-chain and implement a decentralized public consensus mechanism for authorizing changes to the system. These changes can be configuration parameters updates, adding or removing new objects/templates, contingency mechanisms in case of system failures, etc. Users of the system participate in the decision-making process by staking the relevant crypto-asset and proposing changes to the system through smart contracts. The existing users can then vote on these changes within the implemented time-lock threshold, and, if approved, the changes take effect automatically through the execution of the proposed smart contracts. The mechanism allows anyone to stake their crypto assets and have a say in the governance of the system. Every proposed change and vote is recorded on-chain, making it tamper-proof and resistant to manipulation/forgery. The implementation takes inspiration from how governance mechanisms are implemented on popular decentralized finance (DeFi) Daaps on the Ethereum blockchain e.g MakerDAO [23] and Uniswap [24].

```
stake_threshold: 100  // must own minimal stake
max_operator_number: 5000  // number of operators
block_interval: 5s //
epoch_interval: 500 // 500 blocks
...
artifacts: myDappArtifactContract
topology:  myDappTopologyContract
template: myDappTemplateContract
security: myDappSecurityContract
identity: myDappIdentityContract
...
```

Core The core chain code stores common attributes, metadata, and definitions for the sharding network. It can specify the minimal stake required for participation. Here we assume that stake is divided into a unit stake, for instance, 100 stakes per worker node. This means that if a stake owner has 500 stakes, then he/she can contribute five worker nodes. Each worker node is administrated by an operator – similar to the concept of orchestrator. Operators are stake owners. An operator can manage multiple worker nodes. The number can be determined according to the amount of stake owned. In addition, the core contract can keep a list of references that point to other smart contracts.

```
operator_type:
  description: "myDappOperator type"
  derived_from: myDappNode
  requirements:
    minimal_stake: {{core.stake_threshold}}
    minimal_reputation: {{core.min_reputation}}
operators:
  description: "node operators" // a list of candidate operators
  list: \\
   — name: "alice@myDapp"
      type: operator_type
      identity:
        cert:    // X.509 certificate
        type: "X.509"
        stake: 1000
   — name: "bob@myDapp"
      type: operator_type
      identity:
        cert:    // X.509 certificate
        type: "X.509"
        stake: 800
    ...
```

Names and identifiers This contract keeps track of names, identifiers, certificates, and user accounts like qualified operator lists. Many blockchain systems have implemented built-in identity management, which can be leveraged. For Hyperledger Fabric, it applies X.509 based credentials. In the case of Ethereum, wallet addresses can be used. Stake owners who want to host worker nodes can add themselves as operators by sending a request transaction, which will invoke a function to handle it. The request will be verified, for instance, checking whether the requester has a sufficient stake or whether the operator list is full before the stake owner is inserted into the operator list. There are functions defined for adding, removing, and validating operators.

```
vm_worker_image:
  description: "worker node vm image"
  type: myDappArtifact
  properties:
    id: "92a78ef0-31d1-11ec-8d3d-0242ac130003"
    IPFS_CID:
    file_name: "myDappWorkerVM"
    attributes:
      -type: qcow2
      -size: 20G
       ...
    encryption:
      -algorithm: AES
      -key_distribution_method: "myDappSecretSharing"
       ...
```

Artifacts The contract for artifacts records virtual machine or container images and source code repositories like GitHub project links. The actual files are stored off-chain, for example using IPFS [18]. Only metadata is kept on-chain.

```
topology:
node_groups:
  description: "node groups (shards)"
  derived_from: myDappTopology
  properties:
    name: "shard"+{{index}}  // shard name: shard1, ...
    occurrence: 20 // 20 shards
    workers_per_shard: 250
    epoch_interval: {{core.epoch_interval}}
  requirements:
    composed_of: worker_node
worker_node:
  description: "worker node"
  derived_from: myDappNode
  properties:
    occurrence: 5000
    image: {{artifacts.vm_worker_image}}
    ports:
      -target: 12345
       published: 12345
       protocol: TCP
     ...
  requirements:
    operator_selected_from: {{identity.operators}}
     ...
```

Decentralized infrastructure specification and topology The contract is used to specify the envisioned topology in detail, like shards, worker node details (e.g., link to the virtual machine or container image, link to any source codes, network ports, service interfaces, resources desired like GPUs). It defines the number of available worker nodes per shard and the number of shards. It can further specify requirements for the infrastructure like optimization policies. For instance, a policy requires that worker nodes administrated by the same operator be allocated to different shards. Moreover, topology details can be defined using relationships. In addition, validator node types can be specified for verifying status of the worker nodes. A validator node can run pre-defined test requests against a worker node. In validator node type, one can determine the commands for running these tests.

```
node_template:
 - id: ab76141a-31db-11ec-8d3d-0242ac130003
    description: "node template"
    type: worker_node
    properties:
      group: 1
      ipv4_addr:  169.25.138.132
      status:
        -alive: 6/15/2021 13:45:30.617
        -measurement:  // measurement results
    relationship:
      operated_by: "alice@myDapp"
 - id: bdc94556-31db-11ec-8d3d-0242ac130003
    description: "node template"
    type: worker_node
    properties:
      group: 2
      ipv4_addr: 239.245.96.116
      status:
        -alive: 6/15/2021 13:42:21.020
        -measurement:  // measurement results
    relationship:
      operated_by: "bob@myDapp"
  ...
```

Mapper The mapper contract takes decentralized infrastructure specification and data contained in other contract states as input. It computes infrastructure templates that can be converted later to deployable TOSCA templates. One essential function of the mapper contract is to allocate worker nodes to the operators. Other nodes of the blockchain network can verify the allocation plan. For security reasons, allocation is randomized. There are multiple approaches to implement a randomization function on-chain, for instance, using block hash as a random source in PoW based blockchain or using VRF(verifiable random function) [25]. In this work, we apply VRF. Reconfiguration is necessary after updates to other parts of the infrastructure specification. This means that changes to other contract states can trigger the execution of the mapper contract. An alternative approach is that each contract tracks whether its state is dirty – require infrastructure reconfiguration. When a transaction is received from exporting infrastructure templates, it will trigger re-execution of the mapper function that re-configures the infrastructure templates. Orchestrators periodically check the blockchain for infrastructure updates.

```
node_blacklist: [node_uuid, ...]  // list of nodes
operator_blacklist: [operator_name, ...]
permissions:
 - artifacts:
      update_by: admin_user
      admin_users: []
 - core:
     update_by: governance
  ...
...
```

Security policies and configuration setting The contract stores security policies and related settings like maintaining blacklists of operators, worker nodes (using their IP addresses), compromised credentials. It also keeps track of permission lists like who can update certain infrastructure specifications and on-chain states, such as permitted accounts which can update artifacts when new images are released.

After worker nodes are created, orchestrators will upload data like IP addresses using smart contract. Validator nodes can check the worker nodes by sending requests. To verify whether a worker node has minimal computing resources, the validator nodes can send carefully designed tasks to the worker node such as a PoW puzzle. A threshold response time can be used to determine if the worker node has sufficient computing resources. Another approach is to use a reputation score that measures average task completion time for the worker node in the past. A third approach can be applied in case of the TEEs where TEE measurement reports can include information like CPU model and hardware setting of a worker node.

2) Decentralized Computing Network Using TEEs: The second example is a TEE based outsourced computing environment for data processing and machine learning applications. Similar decentralized infrastructures have been proposed in the literature for running AI applications. The

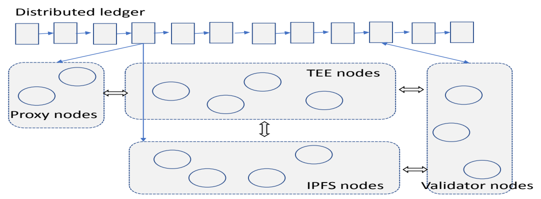


Figure 3: A decentralized computing infrastructure with TEE worker nodes and other nodes with different roles (validators, decentralized storage nodes, SGX proxy nodes).

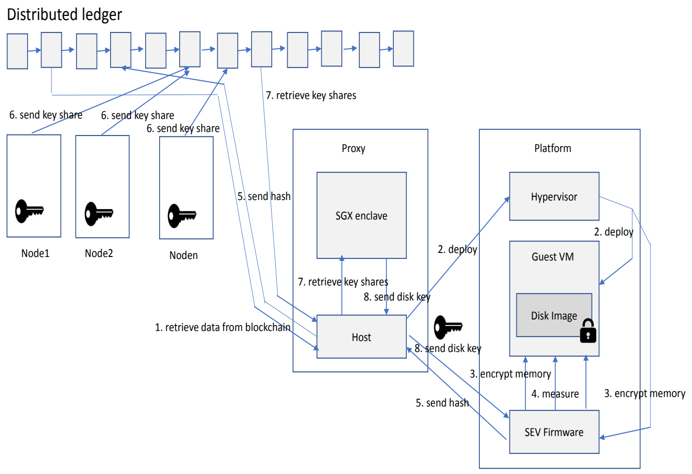


Figure 4: TEE worker node and disk decryption key distribution using blockchain facilitated secret sharing.

IPFS network stores encrypted disk images where data to be processed is kept. Worker nodes apply the TEEs for protecting data confidentiality when the data is processed. To simplify the discussion, we use AMD SEV TEEs [26] that provide out-of-box support for encrypted virtual machines. AMD TEEs implement memory encryption and protect virtual machines from the compromised hypervisor or localhost [27], [28], [29]. In this work, we do not consider TEE attacks such as side-channels which are outside the scope of this paper. We assume that the TEEs are secure enough for use cases of confidential computing. TEEs have been proposed for confidential smart contract execution (e.g., [30]).

```
node_template:
 — id: ac81a42a—71db—55ea—7a3e—1b46ae36010a
    description: "proxy template"
    type: proxy_node
    properties:
      ipv4_addr: 169.25.118.12
      image: {{artifacts.vm_proxy_image}}
      enclave: {{artifacts.sgx_proxy_enclave}}
      credential:  // private key generated by enclave
        cert:    // X.509 certificate
        type: "X.509"
      cpu_arch:
        —vendor: "Intel"
        —model: "i7—8705G"
        —sgx:
          sgx1: true
          sgx2: false
          sgx_enclv: false
      status:
          —alive: 6/11/2021 09:15:24.401
          —measurement: //sgx remote attestation result
      relationship:
          operated_by: "carol@myDapp"
          connect_to: //AMD SEV virtual machine (id list)
            —2621acf8—338d—11ec—8d3d—0242ac130003
            —2621af14—338d—11ec—8d3d—0242ac130003
            —2621b00e—338d—11ec—8d3d—0242ac130003
            ...
```

Here we focus on the unique aspects of infrastructure specification for this use case. The contract code can define worker nodes with TEE properties in the infrastructure specification, like providing a list of optional TEE CPU models. After a worker node is launched, the system will verify that the virtual machine is launched under AMD SEV, a functionality supported by the AMD remote attestation protocol. This is achieved by submitting the measurement report. To work with the AMD TEE and remote attestation protocol out-of-box, we introduce a SGX based proxy as a relay of disk decryption keys. This eliminates the dependence of a single party for holding disk decryption keys. The design applies secret sharing [31]. Shares of a disk decryption key are distributed to multiple entities in the system. After receiving the measurement report and verifying that a virtual machine instance is properly launched by an AMD SP using SEV memory encryption, encrypted key shares will be uploaded to the blockchain. The key shares can be retrieved and sent to the SGX proxy, where the shares will be assembled into the disk decryption key (k out of n shares is enough). The SGX enclave uses a secure communication channel with the AMD SP, created according to the AMD protocol. The SGX proxy uses this channel to send the assembled key that will be forwarded to the virtual machine instance. After receiving the key, the encrypted disk can be decrypted. Then the confidential virtual machine instance hosted under AMD SEV can process the decrypted data. It is worth mentioning that the idea of enclave proxy/relay has been implemented in systems like ARM Veracruz where a SGX proxy server is used for supporting cross-platform remote attestation in a heterogeneous TEE environment. A difference is that in our example use case, SGX proxy is used for key distribution.

To summarize, the extended TOSCA using smart contracts is a universal tool and standard-based approach for describing decentralized infrastructures to support a wide range of Dapp application scenarios.

V. Security Analysis

Security vulnerabilities can be categorized based on the software stack of a blockchain-based system. As discussed earlier, we assume that the blockchain system is properly implemented and defense mechanisms are in place to mitigate security risks at the network, blockchain, and smart contract layer. The focus of the security risk analysis is on vulnerabilities specific to our design. Furthermore, we do not consider attacks particular to a Dapp use case built on top of our system. Other attacks such as exploits of TEE vulnerabilities are also outside the scope of this work [32].

Tamper of smart contracts Attacks can be launched targeting the smart contract programs used for managing decentralized infrastructures. We assume that the smart contract language itself is secure. Transactions that reconfigure a decentralized infrastructure need to be verified on-chain before taking effect. Only authorized entities can make changes to the infrastructure specifications, security policies, metadata, etc. A multi-signature-based approach is used to update smart contracts and on-chain states to handle the risk of key loss or compromised private keys.(e.g., [33]). The blockchain itself provides security assurance like immutability and integrity for the infrastructure data that is stored on-chain.

Cheating of resources provided A worker node can cheat by not providing resources. As we have already discussed, there are multiple approaches to mitigate such risk. Firstly,

Table III: Summary of attack surface and mitigation strategies.

Layers	Threats	Analyses
Application use cases	Use case specific attacks like fairness, verification of computing results, load balancing	Addressed by use case specific defense mechanisms.
	Attacks to the TEEs including side-channel exploits	Addressed by approaches mitigating TEE security vulerabilities like security patches, side-channel resistant libraries, address space randomization, etc.
Our protocol & processes	Attacks to governance	Addressed by secure on-chain voting designs, delay of voting rights (defense against flash loan based attack).
	Attacks to orchestration or proxy nodes	Addressed by decentralization (e.g., partitioning Dapp infrastructure into independently managed components), on-chain blacklist of compromised nodes.
	Cheating on resources provided or availability	Addressed by having validator nodes and reputation scores.
	Attacks from compromised or malicious participating nodes	Addressed by enforcing minimal stake requirement, on-chain mapping policies (e.g., maximize decentralization), blacklist of offending nodes, reputation scores.
	Hijack smart contract updates	Addressed by enforcing permission rules and different modes for updating smart contracts (e.g., multi-signature control, on-chain permission checking).
Smart contract & Toolchain	Reentrancy Attack,	Addressed in the system by smart contract security protection, model checking, and execution atomicity.
	DoS with unexpected revert	
	Unchecked call return value	
	Call stack depth limit exceeded	
Data	Various (e.g. data integrity, data availability)	Attacks not specific or dependent on our design and use cases, and mostly addressed by the underlying blockchain protocols and implementation.
Blockchain	Various (e.g., selfish-mining, double spending, bribery attack, Sybil attack)	
Network	Various (e.g., DDoS, eclipse)	

validator nodes can check the worker nodes by sending resource-intensive tasks to verify whether the worker node has sufficient resources. Secondly, reputation scores can be used to rank worker nodes based on their past task completion times. Finally, when worker nodes are TEE-based, measurement reports can be certified. The report can include information such as CPU model and hardware settings.

Compromised orchestrators We assume that individual orchestrators can be compromised. To reduce the risk of a single point of failure, the framework relies on decentralized orchestrators. A decentralized infrastructure is maintained by multiple independent and autonomous orchestrators. Security breaches to a part of the infrastructure can be handled by isolating the compromised nodes from the rest of the system, for instance, using a blacklist (protected as on-chain data).

Attack to on-chain governance An adversary may launch attacks against the governance mechanism like skewing voting outcomes using a flash loan-based attack. These attacks are not unique to our framework because on-chain governance is applied in numerous blockchain projects. Therefore, we can apply the same best practice and defense approaches to protect smart contracts and management processes.

Attack to the random source used for allocation Depending on the use cases, specific security risks may arise if an adversary can manipulate the mapping process that allocates operators to the worker nodes. Hence, randomization plays a critical role as a defense. In this case, we assume that VRF itself is secure.

Malicious operators We assume that operators are rational and financially motivated. The framework applies negative incentives to discourage undesired operator behaviors. A project can enforce that an operator must own a specific amount of stake to participate. It assumes that the majority of operators can be trusted.

VI. Implementation and Evaluation

A. Extended TOSCA

TOSCA defines a modeling framework that is very extensible. In our case, many decentralization and blockchain-related concepts like stake, randomization between operators and worker nodes, are handled on-chain. The evolution of the decentralized infrastructure is also coordinated through smart contracts and blockchain transactions. Orchestrators/operators can retrieve infrastructure specifications and configuration as key-value export documents from the blockchain. The infrastructure modeling key-value pairs are assembled into a TOSCA template document that can be parsed and interpreted by a TOSCA interpreter. For prototyping, we leverage an open source TOSCA interpreter. One such tool is the TOSCA parsing engine implemented in the Alien4cloud orchestrator (https://alien4cloud.github.io/). The tool supports TOSCA files in YAML format. Segments of infrastructure declaration in YAML can be exported from the blockchain using the smart contract function. The TOSCA interpreter can take the segments as inputs and synthesize a deployment scenario including a deployment plan from the declaration and specification. During deployment, artifacts will be downloaded from IPFS or source code repositories according to the deployment plan. The deployment plan in JSON format constitutes technical steps for provisioning nodes of a decentralized computing environment. The future extension of the TOSCA interpreter is to support accelerators for computing such as GPUs and FPGAs, and integrate the deployment of hybrid TEE computing environments.

B. Experiments

For experiments, we use Hyperledger Fabric V2.2 [34]. Hyperleger Caliper(https://www.hyperledger.org/use/caliper) is used as the benchmarking tool.

Hyperledger Fabric is used to implement infrastructure contracts (chain code in Hyperledger terminology) and manage infrastructure key-value pairs, which save the inputs for creating a configuration YAML file. To test the system, we apply randomly generated infrastructures specified as key-value pairs. Table IV summarizes the data collected for *Submit Transaction* and *Export Infrastructure* with different infrastructure sizes (measured in size of key-value pairs). The operation of *Submit Transaction* consists of two parts, reading the infrastructure input data (in JSON format in our experiments) and submitting constructed transaction to the Fabric to store on the chain. We collected the total time it takes for this whole process to take place. Similarly, the generation of infrastructure export also consists of two steps. The first step is querying data from the chain and the second step is generating a configuration `.yaml` file from the returned data. For randomly generated infrastructures, the input data sizes used in the experiment range from 0.7KB to 5.4KB. We can observe that time differences of the two operations are small even with the increased infrastructure size. Table V summarizes several important metrics on the operations of *Create Asset* and *Read Asset*. We used Caliper for collecting stats for the network. We observe that CPU usage for peers was around 14% and reaching to a Max value of 25%. For orderer nodes, we see a Mac of 10% and an average of 5%. For the blockchain network in our experiment, we saw an average of 100MB for peer nodes and 50MB for orderer nodes in terms of average memory utilization. Traffic In/Out for a peer node is around 11/8 (MB), and around 7/13 (MB) for an orderer node. In terms of disk reads, the peer nodes barely reach 1MB. But for disk write we see that peers had 8MB. For orderer nodes, the disk write and disk read is around 13 MB and 4 MB respectively.

Table IV: Performance of creating on-chain infrastructure data (import) and generating infrastructure file (export).

File Size(KB)	Import infrastructure data (sec)	Generate infrastructure file - export (sec)
0.743	3.347	1.269
1.1	3.349	1.237
2.59	3.383	1.266
3.3	3.388	1.247
4.3	3.42	1.250
5.43	3.481	1.292

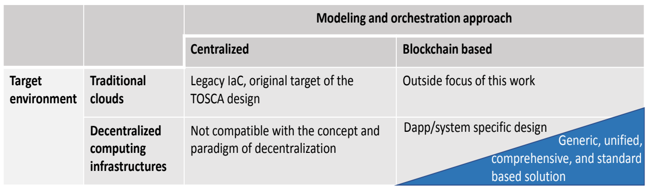
Table V: Performance of asset creating and reading.

Metric	Create Asset	Read Asset
Successes/ Fail	1000/0	10338/0
Send Rate(TPS)	60.5	11.52
Max/Min Latency(s)	11.52/0.46	0.19/0.001
Avg Latency(s)	6.95	45.4
Throughput(TPS)	45.4	175.0

It is possible to further fine-tune Fabric to improve the performance of DAI, which is orthogonal to the scope of this work. For instance, by removing certain bottlenecks like parallelizing endorsement policy verification and optimizing CouchDB bulk read/write, the overall throughput of Fabric can achieve 2,250 tps [35]. FastFabric achieved a throughput of 20,000 tps by reducing computation and I/O overhead during transaction ordering and validation phases [36]. These optimizations do not require any interface changes to the Hyperledger Fabric, and can be easily incorporated into the implementation of DAI.

VII. Related Works

While there has been a considerable amount of work in the field of centralized computing and application of infrastructure-as-code for automation and DevOps, there has not been much consideration for Dapps and decentralized computing environments enabled by the blockchain technology and the associated economic models. Infrastructure-as-code (IaC) is the practice to automatically configure system dependencies and to provide local and remote instances. To harness infrastructure-as-code, administrators write a blueprint that contains deployment specifications ready for orchestration in the clouds. A systematic study [37] on the existing IaC research shows the work that has been done in centralized infrastructure. Hummer et al [38] proposes a testing based infrastructure. There has been research, applying blockchains for managing various facets of the cloud. A survey of blockchain and cloud is provided in [39]. These efforts significantly differ from our work because we target decentralized computing infrastructures and their needs for infrastructure management and automation (using smart contracts) instead of centrally managed cloud-based deployment.



		Modeling and orchestration approach	
		Centralized	Blockchain based
Target environment	Traditional clouds	Legacy IaC, original target of the TOSCA design	Outside focus of this work
	Decentralized computing infrastructures	Not compatible with the concept and paradigm of decentralization	Dapp/system specific design; Generic, unified, comprehensive, and standard based solution

Figure 5: Comparison of DAI with other related works (scope of this work in blue background).

There is plenty of research relying on the TOSCA standards as we see in the work by Artac et al [40]. Similar to the approach in this paper and on which the extension has been proposed. In their work, Tsagkaropoulos et al [41] present a set of TOSCA extensions to model edge computing applications. However, these TOSCA extension efforts focus on other centrally managed ICT infrastructure use cases like support for edge computing. None of them provide a unified approach and standard based modeling tool for decentralized infrastructures and Dapps as described in this paper.

Individual Dapp projects may adopt a customized approach for managing their computing environment like project-specific governance using a smart contract. Our work distinguishes from such effort because we aim at a generic, comprehensive, unified, and standard based approach for modeling, provisioning, and managing decentralized computing infrastructures and Dapps rather than a makeshift design only applicable to a specific system or a single Dapp. We achieve our goal by extending TOSCA standard.

VIII. Conclusion

To develop a unified, general-purpose, comprehensive, and standard-based infrastructure modeling tool for decentralized computing environments and the Dapps empowered by them, we have extended the prominent TOSCA modeling and orchestration language, originally designed for cloud-based deployment, to meet the need of this paradigm shift of computing facilitated by decentralization. Instead of depending on centrally managed components for orchestration, our system leverages smart contracts and the security guarantees of the blockchains for coordinating decentralized infrastructure creation, provisioning, reconfiguration, and evolution. A prototyping environment is created using Fabric to demonstrate this new transformative framework.

References

[1] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, and R. Ranjan, "A taxonomy and survey of cloud resource orchestration techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–41, 2017.

[2] K. Morris, *Infrastructure as code: managing servers in the cloud*. " O'Reilly Media, Inc.", 2016.

[3] A. Brogi, J. Soldani, and P. Wang, "Tosca in a nutshell: Promises and perspectives," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2014, pp. 171–186.

[4] A. Luzar, S. Stanovnik, and M. Cankar, "Examination and comparison of tosca orchestration tools," in *European Conference on Software Architecture*. Springer, 2020, pp. 247–259.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, Dec 2008, accessed: 2015-07-01. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2014, accessed: 2016-08-22. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[7] Golem, "Golem," 2020, available online at: https://golem.network/.

[8] Iexec. Available online at: https://iex.ec/.

[9] "Sonm," available online at: https://sonm.com/.

[10] "Uchain," available online at: https://uchain.world/.

[11] B. Yan, P. Chen, X. Li, and Y. Wang, "Nebula: A blockchain based decentralized sharing computing platform," in *Blockchain and Trustworthy Systems*, Z. Zheng, H.-N. Dai, M. Tang, and X. Chen, Eds. Singapore: Springer Singapore, 2020, pp. 715–731.

[12] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series consensus system," January 2018.

[13] Ocean Protocol Foundation, "Ocean protocol: A decentralized substrate for ai data & services," 2018.

[14] Intel, "Intel software guard extensions," https://software.intel.com/sites/default/files/332680-001.pdf.

[15] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel, and F. Leymann, "A systematic review of cloud modeling languages," *ACM Comput. Surv.*, vol. 51, no. 1, Feb. 2018. [Online]. Available: https://doi.org/10.1145/3150227

[16] K. Kritikos, P. Skrzypek, A. Moga, and O. Matei, "Towards the modelling of hybrid cloud applications," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 291–295.

[17] "Tosca simple profile in yaml version 1.3." [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-ProfileYAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html

[18] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[19] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds. Cham: Springer International Publishing, 2017, pp. 357–388.

[20] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USA: USENIX Association, 1999, p. 173–186.

[21] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[22] H. Team. Harmony whitepaper. version 2.0. Available online at: https://harmony.one/whitepaper.pdf, last accessed on 10.01.2021.

[23] Maker Foundation, "The maker protocol: Makerdao's multi-collateral dai (mcd) system." [Online]. Available: https://makerdao.com/whitepaper/White%20Paper%20-The%20Maker%20Protocol_%20MakerDAO%E2%80%99s%20Multi-Collateral%20Dai%20(MCD)%20System-FINAL-%20021720.pdf

[24] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," Tech. rep., Uniswap, Tech. Rep., 2021.

[25] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pp. 120–130, 1999.

[26] A. SEV-SNP, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, 2020.

[27] D. Kaplan, "AMD x86 memory encryption technologies." Austin, TX: USENIX Association, Aug. 2016.

[28] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," *White paper*, 2016.

[29] "Isolation with integrity protection and more," https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vmisolation-with-integrity-protection-and-more.pdf.

[30] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (Euro S&P)*, 2019, pp. 185–200.

[31] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, ser. SFCS '87. USA: IEEE Computer Society, 1987, p. 427–438. [Online]. Available: https://doi.org/10.1109/SFCS.1987.4

[32] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *ArXiv*, vol. abs/2006.13598, 2020.

[33] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs, "On the security of two-round multi-signatures," Cryptology ePrint Archive, Report 2018/417, 2018, https://ia.cr/2018/417.

[34] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[35] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," *CoRR*, vol. abs/1805.11390, 2018. [Online]. Available: http://arxiv.org/abs/1805.11390

[36] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second," *International Journal of Network Management*, vol. 30, no. 5, p. e2099, 2020, e2099 nem.2099. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2099

[37] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Information and Software Technology*, vol. 108, pp. 65–77, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918302507

[38] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing idempotence for infrastructure as code," in *Middleware 2013*, D. Eyers and K. Schwan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 368–388.

[39] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: A survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 2009–2030, 2020.

[40] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "Devops: Introducing infrastructure-as-code," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 497–498.

[41] A. Tsagkaropoulos, Y. Verginadis, M. Compastié, D. Apostolou, and G. Mentzas, "Extending tosca for edge and fog deployment support," *Electronics*, vol. 10, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/6/737