



Cloud Orchestration at the Level of Application

Project Acronym: **COLA**

Project Number: **731574**

Programme: **Information and Communication Technologies
Advanced Computing and Cloud Computing**

Topic: **ICT-06-2016 Cloud Computing**

Call Identifier: **H2020-ICT-2016-1**
Funding Scheme: **Innovation Action**

Start date of project: 01/01/2017

Duration: 30 months

Deliverable:

D7.4 Security policy formats specification

Due date of deliverable: 29/06/2018

Actual submission date: 06/07/2018

WPL: Nicolae Paladi

Dissemination Level: PU

Version: 1.7

Status and Change History

Table 1 Status Change History

Status:	Name:	Date:	Signature:
Draft:	A. Michalas and N. Paladi	04/06/2018	Nicolae Paladi, A. Michalas
Reviewed:	G. Pierantoni	05/07/2018	Gabriele Pierantoni
Approved:	Tamas Kiss	06/07/2018	Tamas Kiss

Table 2 Document Change History

Version	Date	Pages	Author	Modification
V0.1	02/02		Hai-Van Dang	Document template
V0.2	05/02		Hai-Van Dang	Add sections 2 – Roles in MiCADO, 3 – security related information in TOSCA, and 4 – required security related policies
	27/02	9-16	Antonis Michalas	Change on section 2, 3, 4
	01/03	13-16	Hai-Van Dang	Change on section 4
V0.25	05/03	17	Nicolae Paladi	Initial structure for application security features
V0.3	13/03		Hai-Van Dang	Change title of section 4 into Security actions Add section 5 – security policies
V0.4	05/04	23	Hai-Van Dang	Re-structure the document and updated content
V0.5		24	Antonis Michalas	Change on section 2, 3, 4, 5, 6
V0.55	17/4	24	Nicolae Paladi	Update description of roles and security features
V0.6	19/4	24	Hai-Van Dang	Add section 1 and 7. Change on section 6.
V0.65	26/4	26	Hai-Van Dang	Minor changes. Add section 7 - Extension
V0.7	2/5	26	Antonis Michalas	Change on all sections
V0.8	10/5	27	Hai-Van Dang	Updated all sections
V0.9	24/5	38	Balint Kovacs	Changes on section 2, 3, 4, 5, 6
V0.95	28/5	38	Nicolae Paladi	Edited text to improve readability
V1.0	31/5	38	Hai-Van Dang	Updated all sections
V1.1	1/6	38	Antonis Michalas	Updated all sections
V1.2	28/6	40	Nicolae Paladi	Address reviewer comments
V1.3	28/6	40	Nicolae Paladi	Update Figure 6
V1.4	28/6	41	Nicolae Paladi	Final review and update
V1.5	5/7	45	Hai-Van Dang, Balint Kovacs	Update sections 2, 3, 4
V1.6	5/7	47	Nicolae Paladi	Integrate changes, update §5, final edit.
V1.7	6/7	44	Nicolae Paladi	Address final review comments

Acronyms

Table 3 List of acronyms

ADT	Application Description Template
COLA	Cloud Orchestration at the Level of Application
MiCADO	Microservices-based Cloud Application-level Dynamic Orchestrator
SPM	Security Policy Manager
SEA	Security Enforcer Adaptor
CM	Credential Manager
CS	Credential Store
User	MiCADO user

List of Figures and Tables

Figure 1 MiCADO Architecture [8]	9
Figure 2 Administrator launches MiCADO infrastructure	10
Figure 3 Administrator configures security settings	10
Figure 4 MiCADO user deploys application	11
Figure 5 MiCADO user updates security policies	11
Figure 6 Hierarchy of network security policies.....	16
Figure 3 Access policies for security-sensitive assets	31
Figure 4 Application security enforcement flow	37
Figure 5 User authentication enforcement	38
Figure 6 Security components initialization	38

Tables

Table 1 Status Change History	2
Table 2 Document Change History.....	2
Table 3 List of acronyms	3
Table 4 Security features	13
Table 5 Port setting in Application Description Template[2].....	15
Table 6 L7Proxy policy properties	17
Table 7 SMTP Proxy policy properties	18
Table 8 HTTPProxy Properties.....	21
Table 9 HTTP URI Filter Proxy parameters.....	26
Table 10 Description for network security policy in ADT [10]	26
Table 11, Table 11 Internal database credential an Application Description Template.....	28
Table 12, Types of sensitive information	28
Table 13, Properties of sensitive information	30
Table 14, Description for access policy to application sensitive information	31
Table 15, Cloud user credential settings	33
Table 16, Example of user account in the init file	35
Table 17 Dependencies among components in master node	39
Table 18 Dependency between Security Policy Manager and Credential Manager.....	39
Table 19 Overview of password policy parameters.....	40
Table 20 User account lock policy parameters	41
Table 21 Log policy parameters	41
Table 22 Sensitive information access policy parameters	41

D7.4 Security policy formats specification

Table 23 Database confirmation parameters.....	42
Table 24 User reset password policy parameters.....	42

Table of Contents

Status and Change History	2
Acronyms	3
List of Figures and Tables	4
Table of Contents	6
1 Introduction	7
2 MiCADO user roles and security features	9
2.1 Overview of MiCADO architecture	9
2.2 Identified roles in MiCADO	10
2.3 Infrastructure security features	11
2.4 Application security features	12
2.5 Authentication features	12
2.6 Summary of Security Features	13
3 Application security feature setting in the Application Description Template	15
3.1 Port setting	15
3.2 Firewall configuration	16
3.3 Application sensitive information setting	27
3.3.1 Standard description	28
3.3.2 Extended description	29
4 Infrastructure security features setting in the init file	33
4.1 Cloud user credential setting	33
4.2 Port setting	33
4.3 SSL configuration	34
4.4 User accounts	34
5 Security enforcement flow	36
5.1 Security enforcement in MiCADO	36
5.2 Application security features enforcement	36
5.3 User authentication enforcement	37
5.4 Infrastructure security features enforcement	38
5.4.1 Through the use of init file	38
5.4.2 Through the use of command line	40
5.4.3 Extension	40
6 Summary and Conclusions	43
7 References	44

1 Introduction

This deliverable describes the security policy formats specification that aim to provide an overview of how security features are described in the MiCADO infrastructure.

We classify security features based on the requirements towards from the two main roles considered in the project: the administrator and user roles. The administrator is the entity that launches the infrastructure and defines *infrastructure-level* security features. A user is the entity who deploys an application in the launched infrastructure and defines *application-level* security features. For each role, we classify features into *basic* and *advanced*. Basic features must be implemented while advanced features are left for future development.

Next, we describe how basic features can be implemented in the underlying infrastructure. Administrators control infrastructure-level security features which are configured through configuration values in an initial configuration file (*init* file) used to launch the infrastructure. Users control application-level security features which are configured through policies and artefacts in an Application Description Template file describing the deployed application. The Application Description Template is based on the TOSCA policy specification format [2]. Finally, we illustrate a tentative overall process on how the security enablers defined in a previous WP7 Deliverable[1] interact with the rest of the components to successfully deploy the proposed security features.

Objectives

The objectives of this document are as follows:

- Describe the roles and security features present in the MiCADO architecture.
- Describe the format of application-level security feature settings in the Application Description Template.
- Describe the format of infrastructure-level security feature settings in the infrastructure initialization file
- Describe the security enforcement flow in the MiCADO architecture.

Scope

The purpose of this document is to specify the policy formats for security configuration on the infrastructure and on the application levels. The policy format is described through several concrete configuration examples of security enabler features. The examples can be extrapolated to other security enablers and their features, both on the application and infrastructure level.

The purpose of the document *does not* include defining a novel policy specification format. Instead, this work leverages the TOSCA policy specification format. The security policy formats described in this document will be adopted by the MiCADO framework for configuration on the infrastructure and application level.

D7.4 Security policy formats specification

Relation with other work packages and deliverables

This deliverable builds upon and is closely related to several earlier deliverables produced within the COLA project, as follows:

- **WP5 Deliverables:** TOSCA-based Application Description Templates are described in D5.2. The integration of the templates with the selected application description approach is described in D5.3 and the initial set of templates and services of use cases defined in D5.4.
- **WP6 Deliverables:** This deliverable is aligned with the earlier work on the prototype and documentation of the cloud deployment orchestrator service described in D6.1 *Prototype and documentation of the cloud deployment orchestrator service* and D6.2 *Prototype and documentation of the monitoring service*.
- **WP7 Deliverables:** The current deliverable builds upon the earlier work in deliverables D7.1 COLA security requirements and D7.2 *MiCADO security architecture specification*. The current deliverable has been developed concurrently with deliverable D7.3 *MiCADO application security classification specification* and is compatible with the security enabler open specifications described in D7.3.

The current security enabler is intended to provide input to D7.5 *Design and implementations of security modules* and D5.5 *Second set of templates and services of use cases*.

Document structure

The remainder of this document consists of the following chapters:

- *Chapter 2 – Roles and security features:* This chapter identifies the main roles in the infrastructure as well as all the security features that are associated with each role.
- *Chapter 3 – Application security features setting in the Application Description Template:* This chapter illustrates how application security features can be described in an Application Description Template.
- *Chapter 4 – Infrastructure security features setting in init file:* This chapter points out how infrastructure security features can be presented in the init file.
- *Chapter 5 – Security enforcement flow:* This chapter demonstrates a tentative process through which security components can process security features derived from configuration settings that are provided by the administrator or the user during the deployment stage.
- *Chapter 6 – Summary and Conclusion:* This chapter concludes the deliverable.

2 MiCADO user roles and security features

2.1 Overview of MiCADO architecture

Before identifying user roles, we summarize the core architecture of MiCADO described previously in deliverable D6.2 [8]. MiCADO, consists of one master node and an arbitrary set of worker nodes. The master node performs operations related to handling of resources and scheduling of microservices, while the worker nodes execute the actual microservices. Based on the changing requirements of the running microservices, the master node handles the allocation or launch of worker nodes continuously and automatically.

The master node contains five main components: MiCADO Submitter, Cloud Orchestrator, Container Orchestrator, Policy Keeper and Monitoring system. The Optimiser component is an extension later.

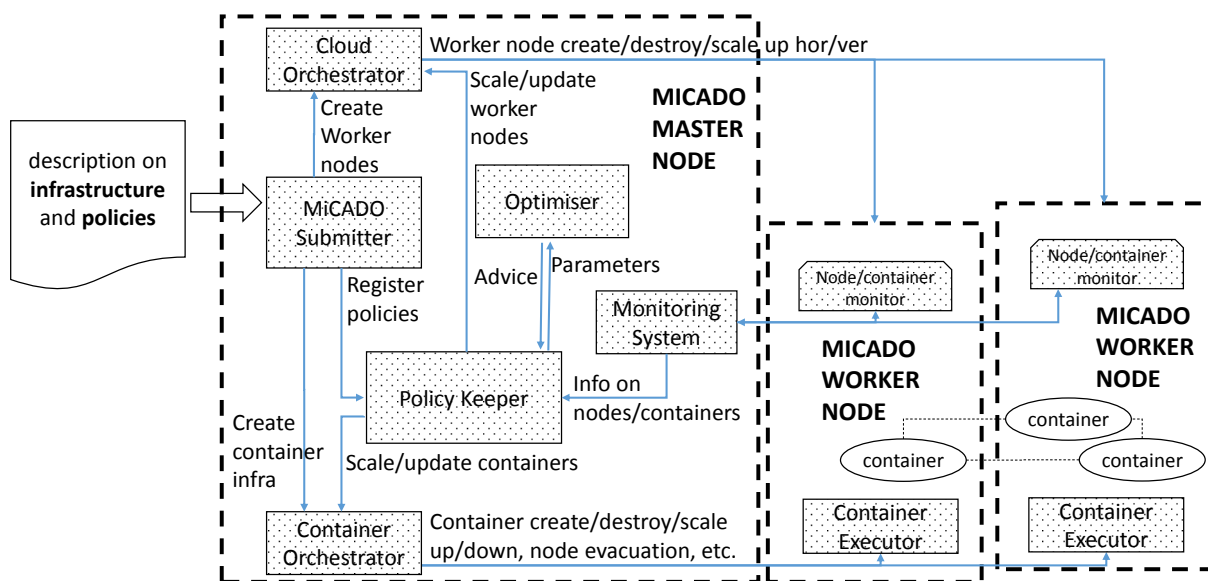


Figure 1 MiCADO Architecture [8]

MiCADO submitter is the entry point where MiCADO users, i.e. users, can input an Application Description Template (ADT) file describing the application topology and the relevant policies into MiCADO. The topology presents components of the application, their Docker images as well as their relationship. Moreover, it describes the virtual machine configuration for worker nodes on which Docker images will be deployed. Meanwhile, policies illustrate the set of rules used throughout the lifecycle of the application, involving scaling policies and security policies. For more details on the ADT file, please refer to deliverable D5.4 [9]. Next, the Cloud Orchestrator and the Container Orchestrator are the two main components for scaling. Cloud Orchestrator, aims to scale up or down virtual machines (VM) while the Container Orchestrator performs for Docker containers. In addition to that, Policy Keeper is responsible for the auto-scaling feature of MiCADO. It relies on scaling policies described in ADT files and actual monitoring information collected from the Monitoring System to make decisions regarding scaling up or down virtual machines and/or containers. Finally, the Monitoring System is implemented to actively request monitoring data about virtual machines, microservices and Docker containers from the existing monitoring agents in the worker nodes.

D7.4 Security policy formats specification

Specific instalment of such core components of the MiCADO master node could be described through an *init* file. After the *init* file is executed, the master node and a default number of worker nodes are set up. When the MiCADO infrastructure is ready, in order to deploy an application in the launched MiCADO, an Application Description Template (ADT) file is sent into the MiCADO Submitter. The Submitter then parses the ADT and sends appropriate information to the master node components.

2.2 Identified roles in MiCADO

Based on how MiCADO is launched and how application is deployed, we identify that there are two main roles in MiCADO. The first role is the administrator who is responsible for enabling the overall service, i.e. launching MiCADO master node through an *init* file. In case the master node is set up in the cloud, the administrator uploads the *init* file to the cloud service provider and requests the launch of a virtual machine with a set of predefined components (See Figure 2).

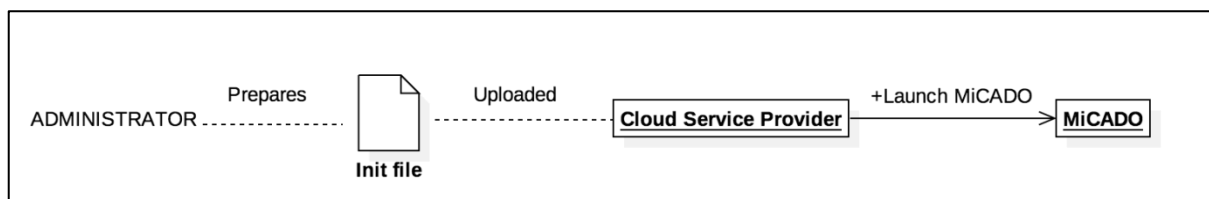


Figure 2 Administrator launches MiCADO infrastructure

In the other scenario, where the administrator initializes the master node in a desktop, the *init* file is executed locally. Later, whenever the administrator wishes to configure the master node security settings, she could perform it by accessing the master node and executing command line instructions (See Figure 3).

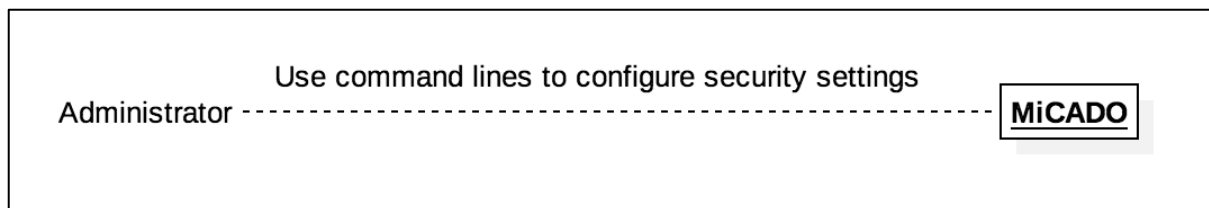


Figure 3 Administrator configures security settings

The second role is the MiCADO user, i.e. user, who may deploy an application in the launched infrastructure. After the successful launch of the infrastructure, the user uploads an *ADT file* to run the application in the infrastructure (See Figure 4). The ADT file describes the application as well as the configuration for the worker nodes on which the application will be deployed. In addition to that, the ADT file also involves security policies relevant to the application, for instance, firewall policies and application sensitive information storing requirements. Later, when the user wishes to update the application security policies, she can upload the updated ADT file (See Figure 5).

D7.4 Security policy formats specification

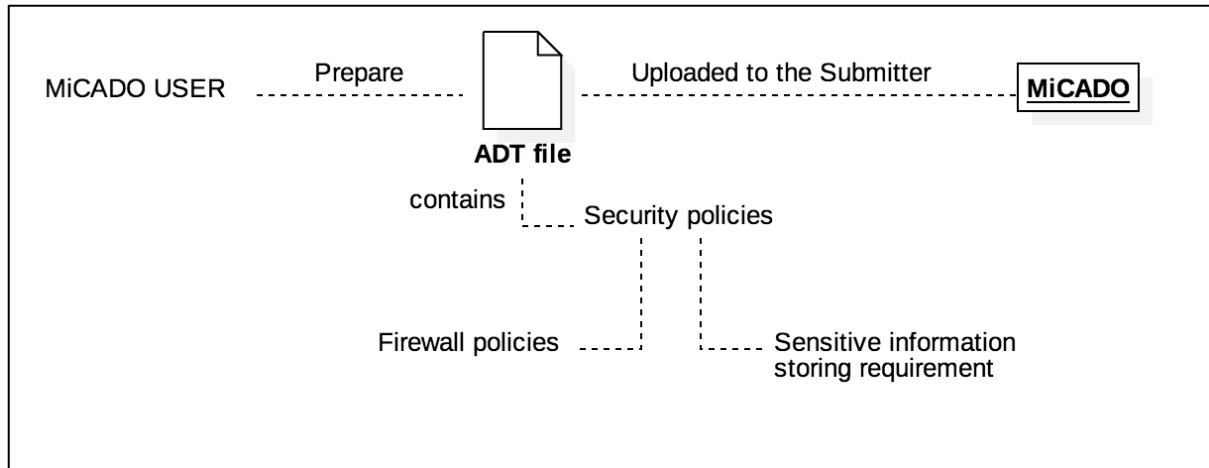


Figure 4 MiCADO user deploys application

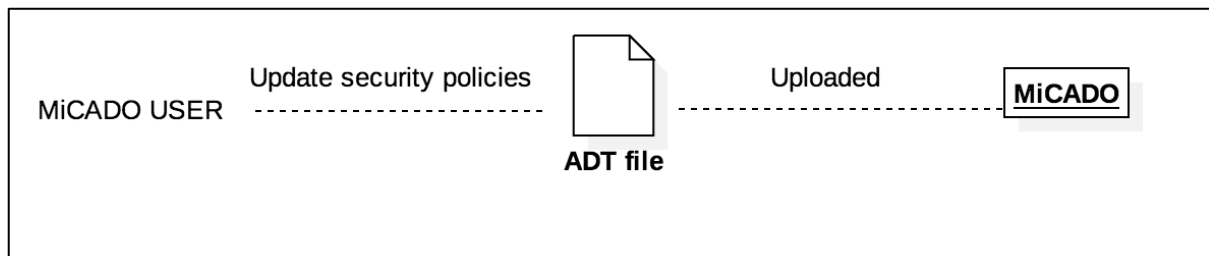


Figure 5 MiCADO user updates security policies

Each role has a set of different security features that needs to be carefully considered and configured. The features that the role administrator is eligible to control are called *Infrastructure Security Features*, while the ones that the role users can control are called *Application Security Features*. Apart from that, by *Authentication Features*, we refer to features related to MiCADO accounts that users can manage. All these features will be thoroughly discussed in the next sections.

2.3 Infrastructure security features

This section aims to identify infrastructure-level security features that the administrator can configure during the launch of MiCADO architecture and re-configure when MiCADO is running. In general, the administrator is eligible to configure security settings for the master node and manage all MiCADO users.

During the setup of the master node, the administrator configures a list of settings such as:

- **Set up SSH to master node.** This allows the administrator to access the master node for further configuration or updates;
- **Add cloud user credentials** for the Cloud Orchestrator component. This enables the Cloud Orchestrator to request the Cloud Service Provider to scale up or down virtual machines;
- **Install digital certificates** (this will allow the protection of the communication between different entities through TLS/SSL) for the master node;
- **Add users' accounts** into MiCADO infrastructure. Such information will be used to authenticate users before allowing them to deploy applications in.

D7.4 Security policy formats specification

Apart from setting up SSH that could be configured through Cloud Service Provider, other features could be configured through the *init* file and/or command lines.

After the infrastructure has been launched, the administrator should be able to perform updates such as:

- Update cloud user credential (extended feature);
- Remove cloud user credential (extended feature);
- Re-configure open ports on master node (extended feature);
- Re-configure network security policies (extended feature);
- Update SSL certificate (extended feature);
- Reset password for a user;
- Add a new user;
- Remove a user;
- Create/Define password rules for users (extended feature).

All these features can be configured through the command line.

2.4 Application security features

This section aims to identify the application-level security features that the MiCADO user can configure during the launch of an application and re-configure when the application is running.

After the successful launch of the infrastructure, the user uploads an ADT file to run the application in the infrastructure. The ADT file describes the application, the configuration for the worker nodes on which the application will be deployed, as well as relevant security policies. For instance, the application may require a number of ports to be exposed on worker nodes. Apart from that, users may wish to store sensitive information inside MiCADO so that it can be reachable by the specified running microservices. By this way, users could avoid storing the sensitive information directly in the source code or the Docker file. The following list covers all such configurations:

- Identify open ports for the application;
- Configure network security policies for open ports;
- Add sensitive information to be stored inside MiCADO and configure which microservices are allowed to access the sensitive information.

When the user wishes to perform certain updates while the application is running in the infrastructure, she makes changes to the ADT file and re-upload it into MiCADO. Such updates include the following:

- Re-configure open ports on worker nodes (extended feature);
- Re-configure network security policies for worker nodes (extended feature);
- Update application's sensitive information (extended feature);
- Remove sensitive information of an application from the infrastructure (extended feature).

2.5 Authentication features

Apart from the application security features that MiCADO user can configure, there are other features related to authentication. Not everyone has the right to upload ADT files. Each user

D7.4 Security policy formats specification

has an account that is used for authentication in MiCADO. While the administrator can manage all these accounts, user are able to manage their personal accounts. For instance:

- Authenticate based on user name and password;
- Change password (extended feature);
- Reset password (extended feature).

These features might be done through the REST APIs provided by MiCADO.

2.6 Summary of Security Features

For a more systematic view, we summarize the mentioned security features of both roles in the following table along with the scope of affect and entry point for each feature.

Table 4 Security features

#	Security features	Scope	Entry point
Infrastructure security features			
1	Set up SSH	Master node	Cloud service provider
2	Add cloud user credentials	Master node	Init file and/or Command line
3	Install SSL certificate	Master node	Init file and/or Command line
4	Add user accounts with default passwords	Master node	Init file and/or Command line
5	Reset password for a user	Master node	Command line
6	Update cloud user credential (extended feature)	Master node	Command line
7	Remove cloud user credential (extended feature)	Master node	Command line
8	Re-configure open ports (extended feature)	Master node	Command line
9	Re-configure network security policies (extended feature)	Master node	Command line
10	Update SSL certificate for the master node (extended feature)	Master node	Command line
11	Add a new user	Master node	Command line
12	Remove a user	Master node	Command line
13	Create/Define password rules for users (extended feature)	Master node	Command line
Application security features			
14	Configure firewall to open defined ports for the application	Worker node	ADT file
15	Configure network security policies	Worker node	ADT file
16	Add application sensitive information to be stored inside the infrastructure and configure which microservices could access it	Master node	ADT file
18	Re-configure open ports (extended feature)	Worker node	ADT file
19	Re-configure network security policies (extended feature)	Worker node	ADT file

D7.4 Security policy formats specification

20	Update/ Delete application sensitive information (extended feature)	Master node	ADT file
	Remove sensitive information of an application (extended feature)	Master node	
	Authentication features		
	Authenticate based on user name and password	Master node	Rest API
	Change password (extended feature)	Master node	Rest API
	Reset password (extended feature)	Master node	Rest API

While the infrastructure security features are executed through the *init* file and/or *command line*, the application security features are done through an *ADT* file. Apart from that, the authentication features are performed through *REST APIs*.

In the next two chapters, we examine how information related to application security features could be represented in an *Application Description Template* and how infrastructure security features could be described in the *init* file. Later, we continue with the presentation of the flows for enforcing these security features. The flows describe how components in MiCADO communicate with each other to set up the security features.

3 Application security feature setting in the Application Description Template

In this chapter, we focus on the basic application security features that involves

- Configure firewall to open defined ports for the application
- Configure network security policies
- Add application sensitive information to be stored inside the infrastructure and configure which microservices could access it

More precisely, we investigate how relevant information of the application (e.g. application's sensitive information and firewall setting information for worker nodes), would be defined in an Application Description Template .

3.1 Port setting

We consider the case where an application requires to open certain ports in worker nodes. In the current Application Description Template, a user may describe which ports are required to be open to run an application. We illustrate this below by using the relevant part from the current ADT file developed by the COLA team (See Table 2)

Table 5 Port setting in Application Description Template[2]

```

t o p o l o g y _ t e m p l a t e :
  i n p u t s :
    p o r t _ e x p o s e d _ d a :
      t y p e : i n t e g e r
      d e s c r i p t i o n : p o r t   e x p o s e d   f o r   d a t a _ a v e n u e
      r e q u i r e d : y e s
      d e f a u l t : 8080
    p o r t _ e x p o s e d _ m y s q l :
      t y p e : i n t e g e r
      d e s c r i p t i o n : p o r t   e x p o s e d   f o r   m y s q l
      r e q u i r e d : y e s
      d e f a u l t : 3306
  n o d e _ t e m p l a t e s :
    d a t a _ a v e n u e :
      t y p e : t o s c a . n o d e s . M C A D O . C o n t a i n e r . A p p l i c a t i o n . D o c k e r
      p r o p e r t i e s :
        e x p o s e d _ p o r t : { g e t _ i n p u t : p o r t _ e x p o s e d _ d a }
    m y s q l :
      t y p e : t o s c a . n o d e s . M C A D O . C o n t a i n e r . A p p l i c a t i o n . D o c k e r
      p r o p e r t i e s :
        e x p o s e d _ p o r t : { g e t _ i n p u t : p o r t _ e x p o s e d _ m y s q l }

```

In this example, it can be seen that the service *data_avenue* requires port 8080 to be exposed and the service *mysql* requires port 3306. Therefore, ports 8080 and 3306 should be open in *all* worker nodes hosting the application. Based on such services' description in Application Description Template, we should extract the exposed port information, then the underlying firewalls should be configured to open all the necessary ports automatically without the user experiencing any disruption or delay in the overall running of the service.

However, each port may be required to be attached to a network security policy. The applied network security policy makes sure that the constraints configured by the user are enforced, such as:

- application protocol (e.g. HTTP);

D7.4 Security policy formats specification

- TLS encryption;
- protocol-specific security features, such permitted methods or url filtering within HTTP.

For instance, users may require port 8080 to be tied with a *HTTP* policy that only accepts request methods of the type GET, PUT or POST with a timeout of 10 seconds and without any encryption.

In such cases, the above representation in ADT (see Table 5) could not illustrate any network security policies attached to open ports. Consequently, we need another way to express both ports to be open and its relevant security network policies in ADT. Details on security network policies and the way to describe them in ADT will be represented in Section 3.2.

3.2 Firewall configuration

Firewall configuration is done automatically through setting exposed ports for a specific application and a set of pre-defined security policies that correspond to application-level filtering rules.

The policy objects are organized in a hierarchical manner and attributes are added to the specific types through inheritance, where each object type will automatically carry forward the attributes of any of their ancestors. The object that exposes the ports also has a “security” parameter that accepts an object derived from the AbstractNetworkSecurityPolicy type. This fits well with the overall design approach of the Application Description Templates whereby Policies can be arranged in a hierarchy akin to that of classes in an Object-Oriented language.

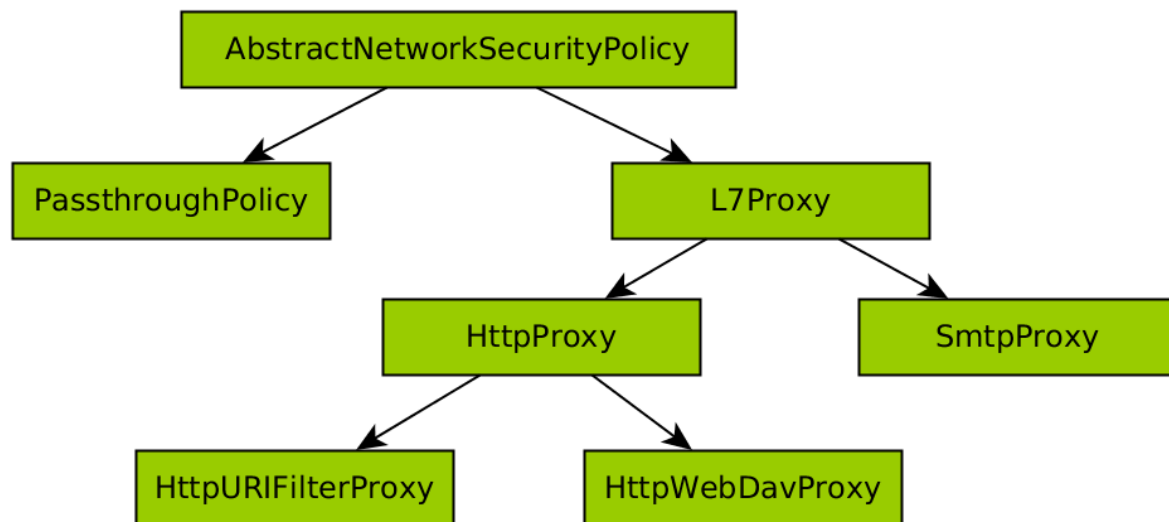


Figure 6 Hierarchy of network security policies

Each Policy in the hierarchy of network security policies (see Figure 6) is further described with two sets of parameters: generic parameters that describe the metadata of the policy and parameters that are specific, these sets of parameters are listed in the following sections. Based on such information, corresponding security policies are defined in Application

D7.4 Security policy formats specification

Description Template [10] to fulfil the requirement of describing both open ports and its relevant network policies in ADT.

AbstractNetworkSecurityPolicy

Description:

- **Name:** toasca.policies.MiCADO.Security.Network
- **Type:** Abstract container of all security policies
- **Description:** Requires to set specific configuration for firewalls in worker nodes

Derived from: toasca.policies.Root

PassthroughPolicy

Description:

- **Name:** toasca.policies.MiCADO.Security.Network.Passthrough
- **Type:** Policy that specifies no filtering
- **Description:** Policy that specifies that no additional filtering should be done and no application-level firewall should be applied on the traffic

Derived from: toasca.policies.MiCADO.Security.Network

L7Proxy

Description:

- **Name:** toasca.policies.MiCADO.Security.Network.L7Proxy
- **Type:** Policy that specifies application level relaying and TLS control
- **Description:** Policy that specifies no additional protocol enforcement, but states that and application-level firewall should be applied to the traffic and also can provide TLS control
- **Target:** Worker nodes

Derived from: toasca.policies.MiCADO.Security.Network

Properties:

Table 6 L7Proxy policy properties

Name	Type	Subtype	Required	Description
target_ports	TOSCA application's exposed port		yes	Target ports that this policy is attached to
Stage	String		yes	Deployment
Priority	Integer		yes	100
Encryption	boolean		yes	Specifies if encryption should be employed for the protocol proxy

D7.4 Security policy formats specification

encryption_key	String		no	The key file to be used for TLS encryption in unencrypted PEM format
encryption_cert	String		no	The certificate file to be used for TLS encryption in PEM format
encryption_offload	String		no	Controls whether the connection should be re-encrypted on the server side
encryption_cipher	String		no	Specifies the allowed ciphers on the client side during TLS handshake

SmtProxy

Description:

- **Name:** tosa.nodes.MiCADO.SecurityPolicy.Network.SmtProxy
- **Type:** Policy that specifies that the SMTP protocol should be enforced
- **Description:** Policy that specifies SMTP protocol enforcement, specifies that an application-level firewall should be applied to the traffic and also can provide TLS control
- **Target:** Worker node

Derived from: tosa.nodes.MiCADO.SecurityPolicy.L7Proxy

Properties:

Table 7 SMTP Proxy policy properties

Name	Type	Subtype	Required	Description
target_ports	TOSCA application's exposed port		yes	Target ports that this policy is attached to
Stage	string		yes	Deployment
Priority	int		yes	100
relay_check	boolean		yes	Enable disable relay checking
permit_percent_hack	boolean		no	Allow the % sign in the local part of e mail addresses
error_soft	boolean		no	Return a soft error condition when recipient filter does not match If enabled the proxy will try to re validate the recipient and send the mail again This option is useful when the server used for the recipient matching is down
relay_domains	list	string	no	Domains mails are accepted for Use Postfix style lists E g example com allows every

D7.4 Security policy formats specification

				subdomain of example com but not example com To match example com use example com
permit_exclamation_mark	boolean		no	Allow the sign in the local part of e mail addresses
relay_domains_matcher_whitelist	list	string	no	Domains mails are accepted for based on a list of regular expressions (has precedence over blacklist)
relay_domains_matcher_blacklist	list	string	no	Domains mails are rejected for based on a list of regular expressions
sender_matcher_whitelist	list	string	no	Sender addresses that are accepted for based on a list of regular expressions (has precedence over blacklist)
sender_matcher_blacklist	list	string	no	Sender addresses that are explicitly rejected for based on a list of regular expressions
recipient_matcher_whitelist	list	string	no	Recipient addresses that are accepted for based on a list of regular expressions (has precedence over blacklist)
recipient_matcher_blacklist	list	string	no	Recipient addresses that are explicitly rejected for based on a list of regular expressions
autodetect_domain_from	enum ("mailname", "fqdn")	string	no	If you want Zorp to autodetect the domain name of the firewall and write it to the Received line then set this This attribute either set the method how the Zorp detect the mailname Only takes effect if add_received_header is TRUE
append_domain	string		no	Domain to append to email addresses which do not specify domain name An address is rejected if it does not contain a domain and append_domain is empty
permit_omission_of_angle_brackets	boolean		no	Permit MAIL From and RCPT To parameters without the normally required angle brackets around them They will be added when the message leaves the proxy anyway
interval_transfer_n	int		no	The interval between two

D7.4 Security policy formats specification

oob				NOOP commands sent to the server while waiting for the results of stacked proxies
resolve_host	boolean		no	Resolve the client host from the IP address and add it to the Received line Only takes effect if add_received_header is TRUE
permit_long_responses	boolean		no	Permit overly long responses as some MTAs include variable parts in responses which might get very long If enabled responses longer than
max_auth_request_length	int		no	Maximum allowed length of a request during SASL style authentication
max_response_length	int		no	Maximum allowed line length of a server response
unconnected_response_code	int		no	Error code sent to the client if connecting to the server fails
add_received_header	boolean		no	Add a Received header into the email messages transferred by the proxy
domain_name	string		no	If you want to set a fix domain name into the added Receive line set this Only takes effect if add_received_header is TRUE
tls_passthrough	boolean		no	Change to passthrough mode after a successful STARTTLS request Zorp does not process or change the encrypted traffic in any way it is transported intact between the client and server
Extensions	list	string	no	List of allowed ESMTP extensions, indexed by the extension verb e g ETRN
require_crlf	boolean		no	Specifies whether the proxy should enforce valid CRLF line terminations
Timeout	int		no	Timeout in milliseconds If no packet arrives within this interval the connection is dropped
max_request_length	int		no	Maximum allowed line length of client requests
permit_unknown_command	boolean		no	Enable unknown commands

HttpProxy

Description:

- **Name:** `tosca.nodes.MiCADO.SecurityPolicy.Network.HttpProxy`
- **Type:** Policy that specifies application level relaying and TLS control
- **Description:** Policy that specifies HTTP protocol enforcement and states that and application-level firewall should be applied on the traffic and also can provide TLS control
- **Target:** Worker node

Derived from: `tosca.nodes.MiCADO.SecurityPolicy.L7Proxy`

Properties:

Table 8 HTTPProxy Properties

Name	Type	Subtype	Required	Description
target_ports	TOSCA application's exposed port		yes	Target ports that this policy is attached to
Stage	string		yes	Deployment
Priority	int		yes	100
max_keepalive_requests	int		no	Maximum number of requests allowed in a single session If the number of requests in the session the reaches this limit the connection is terminated The default 0 value allows unlimited number of requests
permit_proxy_requests	boolean		no	Allow proxy type requests in transparent mode
reset_on_close	boolean		no	Whenever the connection is terminated without a proxy generated error message send an RST instead of a normal close Causes some clients to automatically reconnect
permit_unicode_url	boolean		no	Allow unicode characters in URLs encoded as u This is an IIS extension to HTTP UNICODE UTF 7 UTF 8 etc URLs are forbidden by the RFC as default
permit_server_requests	boolean		no	Allow server type requests in non transparent mode

D7.4 Security policy formats specification

max_hostname_length	int		no	Maximum allowed length of the hostname field in URLs
parent_proxy	string		no	The address or hostname of the parent proxy to be connected Either DirectedRouter or InbandRouter has to be used when using parent proxy
permit_ftp_over_http	boolean		no	Allow processing FTP URLs in non transparent mode
parent_proxy_port	int		no	The port of the parent proxy to be connected
permit_http09_responses	boolean		no	Allow server responses to use the limited HTTP 0 9 protocol As these responses carry no control information verifying the validity of the protocol stream is impossible This does not pose a threat to web clients but exploits might pass undetected if this option is enabled for servers It is recommended to turn this option off for protecting servers and only enable it when Zorp is used in front of users
rewrite_host_header	boolean		no	Rewrite the Host header in requests when URL redirection is performed
max_line_length	int		no	Maximum allowed length of lines in requests and responses This value does not affect data transfer as data is transmitted in binary mode
max_chunk_length	int		no	Maximum allowed length of a single chunk when using chunked transfer encoding The default 0 value means that the length of the chunk is not limited
strict_header_checking_action	enum("accept", "drop", "abort")	string	no	This attribute control what will the Zorp do if a non rfc conform or unknown header found in the communication Only the HTTP_HDR_ACCEPT

D7.4 Security policy formats specification

				HTTP_HDR_DROP and HTTP_HDR_ABORT can be used
target_port_range	string		no	List of ports that non transparent requests are allowed to use The default is to allow port 80 and 443 to permit HTTP and HTTPS traffic The latter also requires the CONNECT method to be enabled
strict_header_checking	boolean		no	Require RFC conformant HTTP headers
max_auth_time	int		no	Request password authentication from the client invalidating cached one time passwords If the time specified in seconds in this attribute expires Zorp requests a new authentication from the client browser even if it still has a password cached
max_url_length	int		no	Maximum allowed length of an URL in a request Note that this directly affects forms using the GET method to pass data to CGI scripts
timeout_request	int		no	Time to wait for a request to arrive from the client
rerequest_attempts	int		no	Controls the number of attempts the proxy takes to send the request to the server In case of server failure a reconnection is made and the complete request is repeated along with POST data
error_status	int		no	If an error occurs Zorp uses this value as the status code of the HTTP response it generates
keep_persistent	boolean		no	Try to keep the connection to the client persistent even if the server does not support it
error_files_directory	string		no	Location of HTTP error messages
max_header	int		no	Maximum number of

D7.4 Security policy formats specification

_lines				header lines allowed in a request or response
use_canonicalized_urls	boolean		no	This attribute enables URL canonicalization which means to automatically convert URLs to their canonical form This enhances security but might cause interoperability problems with some applications It is recommended to disable this setting on a per destination basis URL filtering still sees the canonicalized URL but at the end the proxy sends the original URL to the server
max_body_length	int		no	Maximum allowed length of an HTTP request or response body The default 0 value means that the length of the body is not limited
require_host_header	boolean		no	Require the presence of the Host header If set to FALSE the real URL cannot be recovered from certain requests which might cause problems with URL filtering
buffer_size	int		no	Size of the I O buffer used to transfer entity bodies
permitted_responses	list	dict(string, int)	no	Normative policy hash for HTTP responses indexed by the HTTP method and the response code e g PWD 209 etc See also
transparent_mode	boolean		no	Set the operation mode of the proxy to transparent TRUE or non-transparent FALSE
Permit_null_response	boolean		no	Permit RFC incompliant responses with headers not terminated by CRLF and not containing entity body
Language	string		no	Specifies the language of the HTTP error pages displayed to the client English
error_silent	boolean		no	Turns off verbose error

D7.4 Security policy formats specification

				reporting to the HTTP client makes firewall fingerprinting more difficult
permitted_requests	list	string	no	List of permitted HTTP methods, indexed by the HTTP method e.g GET PUT etc
use_default_port_in_transparent_mode	boolean		no	Set the target port to the value of
timeout_response	int		no	Time to wait for the HTTP status line to arrive from the server
permit_invalid_hex_escape	boolean		no	Allow invalid hexadecimal escaping in URLs must be followed by two hexadecimal digits
auth_cache_time	int		no	Caching authentication information this amount of seconds
Timeout	int		no	General I O timeout in milliseconds If there is no timeout specified for a given operation this value is used
default_port	int		no	This value is used in non-transparent mode when the requested URL does not contain a port number The default should be 80 otherwise the proxy may not function properly

HttpURIFilterProxy

Description:

- **Name:** tosca.nodes.MiCADO.SecurityPolicy.Network.HttpURIFilterProxy
- **Type:** Policy that specifies that the HTTP protocol should be enforced and provides URL filtering
- **Description:** Policy that specifies HTTP protocol enforcement with regex-based URL filtering capabilities, specifies that an application-level firewall should be applied to the traffic and also can provide TLS control
- **Target:** Worker node

Derived from: tosca.nodes.MiCADO.SecurityPolicy.HttpProxy

Properties:

D7.4 Security policy formats specification

Table 9 HTTP URI Filter Proxy parameters

Name	Type	Subtype	Required	Description
target_ports	TOSCA application's exposed port		yes	Target ports that this policy is attached to
Stage	string		yes	Deployment
Priority	int		yes	100
matcher_whitelist	list	string	yes	List of regular expressions determining whether access to an URL is permitted (has precedence over blacklist)
matcher_blacklist	list	string	yes	List of regular expressions determining whether access to an URL is prohibited

HttpWebdavProxy

Description:

- **Name:** tosca.nodes.MiCADO.SecurityPolicy.Network.HttpWebdavProxy
- **Type:** Policy that specifies that the HTTP protocol should be enforced and allows request methods required for WebDAV
- **Description:** Policy that specifies HTTP protocol enforcement with extended set of request methods, but states that an application-level firewall should be applied to the traffic and also can provide TLS control

Next, we use the example in Table 10 to demonstrate how the defined security network policies in ADTs could provide information regarding the ports that are required to be open as well as on the attached network policies.

Table 10 Description for network security policy in ADT [10]

<p>policy_types:</p> <p>tosca.policies.MiCADO.Security.Network:</p> <p>derived_from: tosca.policies.Root</p> <p>description: Base policy for MiCADO network security policies</p> <p>properties:</p> <p>priority:</p> <p>type: integer</p> <p>required: true</p> <p>default: 100</p> <p>stage:</p> <p>type: string</p> <p>required: true</p> <p>default: deployment</p> <p>target_ports:</p> <p>type: list</p> <p>required: true</p> <p>tosca.policies.MiCADO.Security.Network.L7Proxy:</p> <p>derived_from: tosca.policies.MiCADO.Security.Network</p> <p>description: No protocol enforcement. Apply application-level firewall; can provide TLS control</p> <p>properties:</p>
--

D7.4 Security policy formats specification

```

encryption:
  type: boolean
  description: Specifies if encryption should be used
  required: true
encryption_key:
  type: string
  description: The key file for TLS encryption as unencrypted .PEM
  required: false
encryption_cert:
  type: string
  description: The cert file for TLS encryption as .PEM
  required: false
encryption_offload:
  type: string
  description: Controls whether connection should be re-encrypted server side
  required: false
encryption_cipher:
  type: string
  description: Specifies allowed ciphers client side during TLS handshake
  required: false

```

The example describes L7Proxy policy with some defined properties. To identify which ports are attached with this policy, MiCADO user only needs to provide values for the properties *target_ports*.

3.3 Application sensitive information setting

There are various types of sensitive information that applications may require to access during run time. For instance, it is common for applications to have access to a wide variety of databases and storage resources. In such case, the application needs to know the corresponding database/storage credentials. However, this information should not be hard-coded directly into the application's source code because anyone with access to the source code can retrieve this sensitive information. In addition to that, it should not be stored in the docker image because users with access to the actual image can also access that content. For these reasons, it is important to provide users with an option to store such information directly in the infrastructure.

The standard description allows the user to add sensitive information through defining inputs as supported by TOSCA. This method allows the user to provide sensitive information only when submitting the ADT to MiCADO and pass the values for inputs as parameters. This lets the user avoid saving sensitive information inside ADT, which may be published in a source code repository or CI system.

However, the use of the TOSCA construct input does not allow to define additional information which describes the modality with which this information is managed by the infrastructure submission. To achieve this goal we have created a set of policies which can be used to describe in detail not only the sensitive information itself but also the details of how the infrastructure manages them.

3.3.1 Standard description

This use of the TOSCA input construct requires to include the sensitive information in an Application Description Template. In the following example, we show how this can be done [2]:

Table 11, Table 11 Internal database credential an Application Description Template

```

topology_template:
  inputs:
    mysql_database:
      type: string
      description: environment variable to set the database name
      required: yes
    mysql_user:
      type: string
      description: environment variable to set the use name
      required: yes
    mysql_password:
      type: string
      description: environment variable to set the password
      required: yes
  node_templates:
    data_avenue:
      type: tosca.nodes.MCADO.Container.Application.Docker
      requirements:
        - service:
            node: mysql
            relationship: tosca.relationship.ConnectsTo
      mysql:
        type: tosca.nodes.MCADO.Container.Application.Docker
        properties:
          env:
            MYSQL_ROOT_PASSWORD: { get_input: mysql_root_password }
            MYSQL_DATABASE: { get_input: mysql_database }
            MYSQL_USER: { get_input: mysql_user }
            MYSQL_PASSWORD: { get_input: mysql_password }
            exposed_port: { get_input: port_exposed_mysql }

```

In this example, the application *data_avenue* requires to use the database *mysql_database* in *mysql* service. Both the application and *mysql* are deployed as docker services in the infrastructure. However, in order to access the database, the application needs to know the credentials for the database (i.e. *mysql_user* and *mysql_password*).

To summarize, in Table 5 we present all the described security-related information as well as the node types that can be used to describe them in the Application Description Template.

Table 12, Types of sensitive information

#	Parameter	Explanation
	<i>Database credential for internal database service</i>	
1	database_name	Name of database instance

D7.4 Security policy formats specification

2	database_user	Name of user
3	database_password	Password of user
4	database_port	Port of database instance
	<i>Database credential for external database</i>	
6	admin_user	Root user
7	admin_password	Root password of database server
8	url_path	URL path of database, including port
	<i>Username and password credential</i>	
9	user_name	User name
10	password	password
	<i>HTTP basic access authentication credential</i>	
11	token	User name and password that are combined into a string
	<i>X-Auth-Token credential</i>	
12	Token	Token that is encoded in Base64
	<i>OAuth bearer token credential</i>	
13	Token	Token that is encoded in Base64
	<i>OpenStack SSH key pair</i>	
14	Token	A reference (ID) to an existing keypair (already installed)

3.3.2 Extended description

The option to store applications' sensitive information in the infrastructure *should not be configured implicitly*. The main reason for this, is due to the fact that it is infeasible to automatically distinguish which information is sensitive and which is not. Instead, this is something that must be decided by the actual user. Apart from that, this feature should be optional, meaning that the user may choose to store or not such information.

In addition to that, in the MiCADO infrastructure, we aim to provide as much flexibility and maintainability as possible. At the first stage, we would utilize the Container Orchestrator (i.e. docker swarm), to provide secure storage for sensitive information. In such case, we can take advantage of the fact that sensitive information provisioning is managed by Swarm [3]. However, we would possibly provide another option for the secure storage based on Credential Store [1] as an extension. To achieve such flexibility, the system should offer users the option to select one of the existing secure storage types. Therefore, we should design the description of sensitive information in the Application Description Template in such a way that can be easily extended later.

D7.4 Security policy formats specification

Furthermore, an application that is deployed in MiCADO infrastructure could be composed of multiple services. It is common that not all services need to access some specific sensitive information. For instance, assuming that a web application is composed of data controller service, web interface service and business controller service. Only the data controller service requires to access database credential that is sensitive information. Therefore, it is better to let the user limit access to database credential so that only data controller services can access it.

To implement these requirements a family of security policies needs to be established that define pieces of sensitive information and tie into the existing application elements. The policy hierarchy is set up in a way that it is extensible to new types of sensitive information that might be identified in future.

Based on the above, we define the following possible properties of sensitive information that need to be described in the Application Description Template.

Description:

- **Name:** Application sensitive information storage
- **Type:** Application sensitive information storage
- **Description:** Requires application sensitive information to be stored inside MiCADO's master node in such a way that application's docker service can access it at runtime

Properties:

Table 13, Properties of sensitive information

Name	Type	Description
Target	TOSCA node type	Docker service
Stage	String	Deployment
Priority	Integer	100
Secret_name_list	List of string	List of secret names
Secret_value_list	List of string	List of secret values
Secret_storage_list	List of binary	List of secret storage 0 = Docker storage in Swarm (Default value) 1 = MiCADO's credentials store (for extension in the future)
Secret_access_list	Nested list of docker services	For each secret, defining names of docker services which are allowed to access it.

Example:

Assuming that the application is composed of three docker services A, B and C.

```
Secret_name_list = { "db_name", "db_user", "db_pass", "db_port" }
Secret_value_list = { "organization", "test", "123", "5000" }
Secret_storage_list = { 0, 0, 0, 0 }
Secret_access_list = { { A, B }, { A, B }, { A, B }, { A, B } }
```

The policy defines that the user wishes to store 4 pieces of sensitive information as docker secrets, including db_name, db_user, db_pass and db_port. Their corresponding values are

D7.4 Security policy formats specification

organization, test, 123 and 5000. In addition, only services A and B are allowed to access this information.

Such information can be presented as security policy in an Application Description Template as shown in Figure 15. A sample security policy is implemented in [6] and presented in Figure 3.

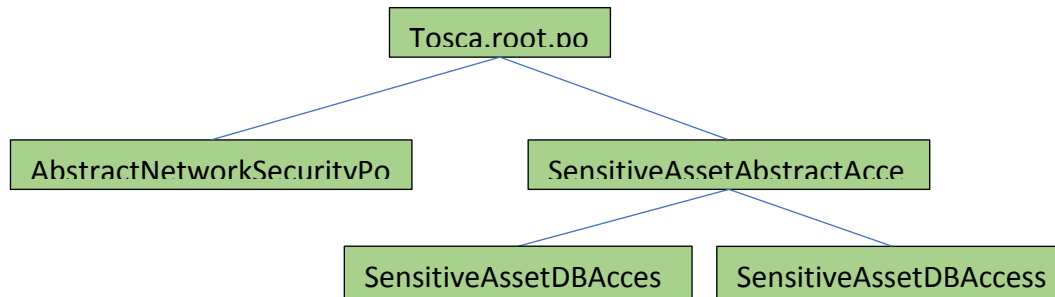


Figure 7 Access policies for security-sensitive assets

Table 14, Description for access policy to application sensitive information

```

inputs:
  my_first_secret:
    type: string
    description: password, for example
    default: none

node_templates:
  app:
    type: tosca.nodes.MCADO.Container.Application.Docker
    properties:
      command: echo "Hello World"
      ports:
        - 80:80
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: busybox
        repository: docker_hub
    requirements:
      - service:
          node: db
          relationship:
            type: tosca.relationships.ConnectsTo
            properties:
              network: default

  db:
    type: tosca.nodes.MCADO.Container.Application.Docker
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: redis
        repository: docker_hub

policies:
  - secret_distribution:
      type: tosca.policies.DockerSecretDistribution
      targets: [ db ]
  
```

D7.4 Security policy formats specification

```

properties:
  stage: execution
  priority: 100
  file_secrets:
    my_pem_secret: tmp/my_file.pem
  text_secrets:
    my_db_name_secret: "name of my db"
- secret_distribution:
  type: tosca.policies.MiCADOSecretDistribution
  targets: [ app ]
  properties:
    stage: execution
    priority: 100
    text_secrets:
      my_other_secret: { get_input: my_first_secret }

```

In this example, the user defines two access policies for the application's sensitive information.

- First policy, defines that only the application 'db' can access the sensitive information retrieved from the file 'tmp/my_file.pem' or from text "name of my db". Furthermore, this sensitive information would be stored as a docker secret in swarm based on the TOSCA type 'tosca.policies.DockerSecretDistribution'.
- Second policy, defines that only the application 'app' can access the sensitive information 'my_first_secret', that would be stored in the Credential Store based on the TOSCA type 'tosca.policies.MiCADOSecretDistribution'.

With this approach, to support future extension of the infrastructure to support various secure storage for different kinds of sensitive information, it would suffice to define a new TOSCA type for the new secure storage.

4 Infrastructure security features setting in the init file

Infrastructure security features are currently not implemented based on policies. A policy-based implementation is considered as a future step. Instead, several infrastructure security features would be initialized through the *init* file, some others would be provisioned automatically while the rest would require administrator's configuration using command line through SSH. In this chapter, we illustrate how basic infrastructure security features can be described directly in the *init* file.

The *init* file is a descriptor in YAML[12] format that is consumed by the configuration management tool (cloud-init in the current implementation, which will be exchanged to Ansible[13] in version 4) to perform security-related settings and adding security-related information to the infrastructure upon initial provisioning. The following *init* file examples are subject to change during implementation and have to be included in their current format in the product documentation.

4.1 Cloud user credential setting

In the current version of MiCADO¹, cloud user credentials are described in the *init* file. At first, the cloud user's credentials (i.e. email address and password), are written in a temporary file. Later, when the Cloud Orchestrator component is deployed, it retrieves the credential from that file.

Table 15, Cloud user credential settings

```
write_files:
#USER DATA
- path: /var/lib/micado/occpus/temp_user_data.yaml
  content: |
    user_data:
      auth_data:
        type: cloudsigma
        email: [cloud user name]
        password: [cloud user password]
```

As in the above example, the cloud user's credentials are temporarily stored in a file that the Cloud Orchestrator (Occopus), can access later. However, storing user's credentials in a static file is considered as insecure. Storing them in a secure storage such as the one provided by the Credential Store (CS) component can properly safeguard this information. CS is designed to be deployed as a web service that provides a restful API that allows a user to store sensitive information and will be added in a future release as a security enhancement. The Security Policy Manager (SPM) would call this API to add the corresponding cloud username and password into the CS. As a consequence, it must be ensured that the CS component would be deployed and launched prior to that API call. This may be managed in the docker-compose file that is used to launch components in the master node as docker containers.

4.2 Port setting

The MiCADO infrastructure requires the firewall to open several ports that its components need. The open ports would be set up by default by the infrastructure. However, for later extension, it could be set up by retrieving necessary ports through the *init* files for both the master and the worker nodes.

¹ At the time that this document is written, MiCADO is on Version 3.0.

D7.4 Security policy formats specification

The infrastructure-related port settings do not affect settings performed through the ADT and are solely responsible for the operation of the MiCADO internal infrastructure. Ports defined in the ADT are opened automatically on the worker nodes on deployment of the application.

Below is the list of the required open ports.

- TCP: 22, 53, 80, 443, 2375, 2377, 7946, 8300, 8301, 8302, 8400, 8500, 8600, 9090, 9093, 9095, 9100, 9200
- UDP: 7946, 8301, 8302, 8600

The above list may change accordingly based on MiCADO's infrastructure changes.

4.3 SSL configuration

Communication between master node and worker nodes, among worker nodes, and between MiCADO with users should be protected via SSL/TLS. Meanwhile, the communication between the deployed application and the end users connecting to the application depends on the application developer. It may be protected or not, depending on specific application development.

1. Communication between master node and worker nodes. For instance, for sending logs and operational monitoring information from worker nodes to the master node. This case is managed by the infrastructure itself, based on its internal components and implementation details.
2. Communication between a user and MiCADO. In this case, we can use the self-signed certificate. The *init* file may describe a X509 key/cert for now, or hostname to automatically generate a self-signed certificate inside MiCADO. However, it can possibly be extended to use LetsEncrypt certificate later. Then the *init* file may describe the hostname and MiCADO would automatically connect to LetsEncrypt for auto certificate provisioning. In general, SSL-related information that can be presented in the *init* file involves the following:
 - SSL-cert-provision-type: { *file*, *self-sign*, *letsencrypt* ² }, meaning that SSL certificate type could be a certificate file, or self-signed certificate generated by the infrastructure, or LetsEncrypt certificate;
 - SSL-cert-hostname: { *host name* }
3. Communication between application users with application assuming that the application supports https. This is considered as an advanced feature that may be extended later. Although it is provided by the infrastructure, it is an application security feature, that cannot be decided until the user deploys their application. The needed x509 keys are to be included in the corresponding sections of the configured security policy and described in the Application Description Template.
4. Communication among application containers in worker nodes: it is protected by Swarm overlay network
5. Communication between MiCADO with admin: protected by SSH

4.4 User accounts

MiCADO's current version does not support authentication. This is a new feature that will be added into the infrastructure. In this scope, authentication means verifying any user who

² LetsEncrypt, a free, automated, and open Certificate Authority. <https://letsencrypt.org/>

D7.4 Security policy formats specification

wishes to deploy an application in the infrastructure, and it is based on password verification. Therefore, in order to perform authentication, a database of users' accounts stored in the master node is required. The Credential Manager (CM) component stores users' accounts and is deployed as a web service that provides a restful API to manage users' accounts, including adding a new user. The relation of CM to other security components is described in Deliverable 7.3.

At least one user account needs to be added prior to the deployment of an application. Although the administrator could add a user by logging into the master node and utilize the command line, it is better to provide an option to add a user at the time of launching the infrastructure. The implementation should support adding the user account (i.e. username and password), on initial provisioning through the *init* file. There are several ways to implement this behaviour, a preliminary design is to use the Security Policy Manager (SPM) component as a workflow owner. The SPM calls the restful API of the CM component with the user account as argument to insert it into the CM as below example.

Table 16, Example of user account in the init file

```

credman:
  image: my_docker_registry/credman
  container_name: credman
  expose:
    - 5001

spm
  depends_on:
    - credman
  image: my_docker_registry/spm
  container_name: spm
  expose:
    - 5003
  command: >
    /bin/bash -c "
      ./wait_for_it.sh credman:5001;
      curl credman:5001/v1.0/adduser -X POST --data '{"username":
"user 01", "password": "123", "email": "user01@mail.com"}'
      python ./my_script.py
    "

```

In the example above, there are two containers, defining security enablers described in Deliverable 7.3:

1. **credman**, implementing the Credential Manager;
2. **spm** implementing the Security Policy Manager.

The Security Policy Manager requests a REST call from the credential manager to add a user into the infrastructure.

The call may have the following form: ' **credman: 5001/ v1. 0/ adduser** '

Note that this is an example and the final implementation may change later.

5 Security enforcement flow

In this section, we present the flow we designed for enforcing the security features. In addition to the application and infrastructure security features, we also elaborate on the adopted authentication feature that verifies users prior allowing them to deploy an application in the infrastructure.

5.1 Security enforcement in MiCADO

In the context of this document, a *security enforcement flow* is the chaining of security components that ensures that the security of data throughout the system. In particular, such chaining prevents unauthorized transfer of data across security domains and restricts transfer to specific interfaces.

The security enforcement flows described in this section aim to preserve the integrity and authenticity of MiCADO user actions (to e.g. authenticate and deploy applications) and administrators (to configure and initialize security components). The hierarchy of security enforcement flows in MiCADO mirrors the hierarchy of existing MiCADO roles (administrator and user).

Users interact with the MiCADO framework primarily to deploy application and describe security policies in ADTs. Administrators describe configurations of security components in *init* files and command line instructions. Note that the purpose of the security enforcement is to configure the correct use of the security enablers, rather than protect the security of specific instances of ADTs. Specific ADT artefacts are protected using industry-standard communication security protocols, such as Transport Layer Security[11].

The security enforcement flows described below implement the MiCADO security architecture described in Deliverable 7.2[1] and will be implemented as part of the upcoming deliverables. While the implementation details are subject to change, the prototypes will be implemented considering the security enforcement flows described below.

5.2 Application security features enforcement

The following figure illustrates a first design of the flow of information that connects the security policies defined in the Application Description Template to the security components.

D7.4 Security policy formats specification

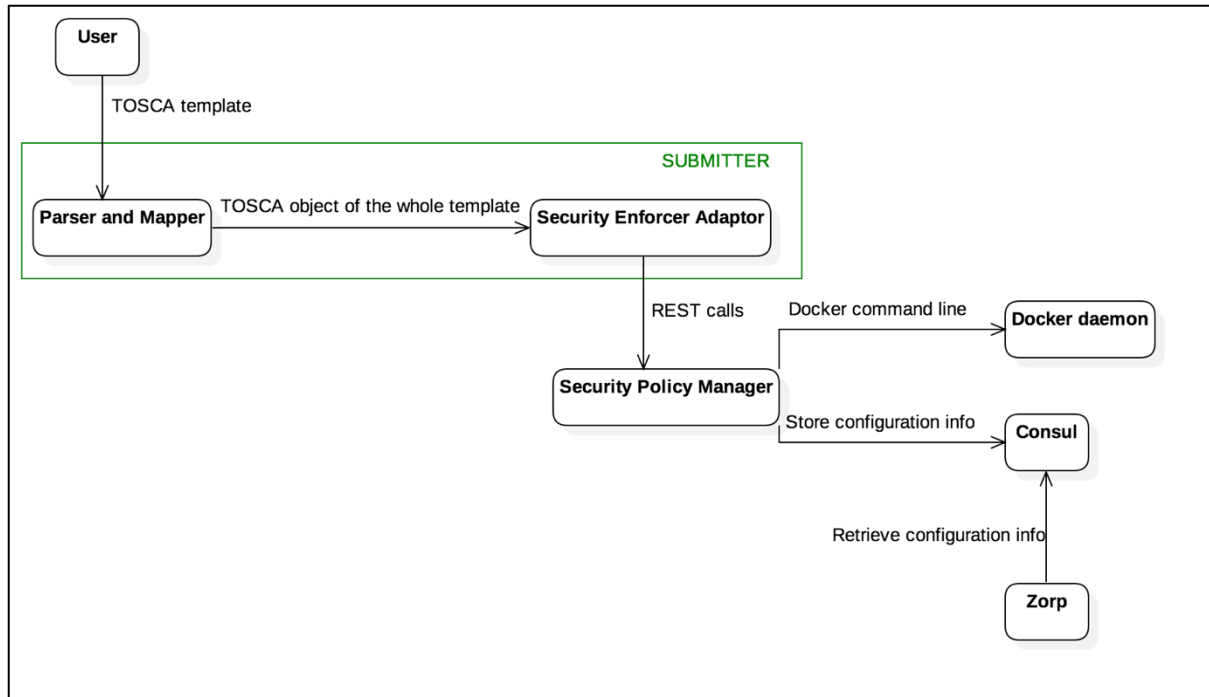


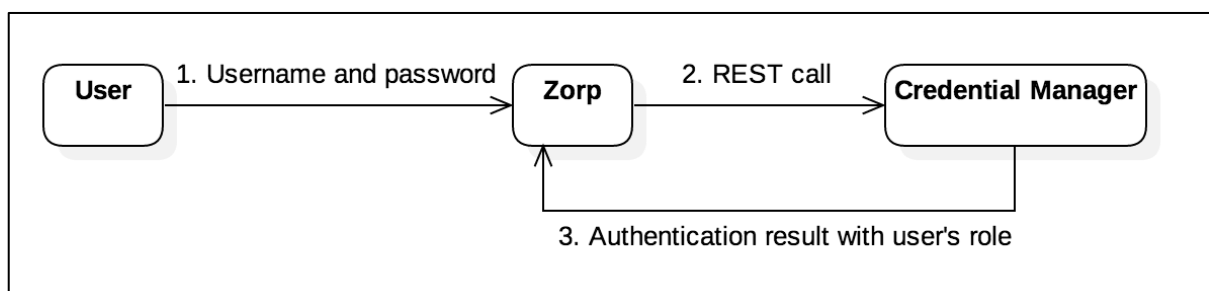
Figure 8 Application security enforcement flow

Description:

1. User uploads ADT file to the Submitter;
2. Upon reception, the sub-component Parser and Mapper of the Submitter parses and maps the file content into a parsed object (a complex Python object returned by the OpenStack TOSCA Parser [7] - whose attributes and methods facilitate future processing of the template) and sends it to the Security Enforcer Adaptor (SEA), that is also a sub-component of the Submitter;
3. SEA translates the TOSCA objects into configuration for Zorp and Docker daemon. The configuration for Docker daemon (the sensitive information storage setting), would contain a list of sensitive information with their required storage and application services that would use them. The configuration for Zorp is based on configured network security policies
4. SEA executes REST calls to Security Policy Manager (SPM);
5. SPM executes a command line call to docker daemon to add sensitive information into swarm and allow appropriate application services to access it;
6. SPM stores configuration information for Zorp into Consul;
7. Zorp retrieves the configuration information from Consul and executes it.

5.3 User authentication enforcement

Figure below, describes how MiCADO executes user authentication.



D7.4 Security policy formats specification

Figure 9 User authentication enforcement

1. User provides user name and password to the infrastructure through protected channel;
2. Upon reception, Zorp component in the infrastructure calls REST API of CM to verify user name and password;
3. CM verifies the received user name and password based on a local file or database that store users's credentials, then returns the result to Zorp;
4. Upon reception, Zorp allows or disallows the user to use the infrastructure based on the received result.

5.4 Infrastructure security features enforcement

5.4.1 Through the use of init file

The following figure depicts what security components are deployed during the launch time of the infrastructure. The arrow between any two components inside the master node indicates their start-up order.

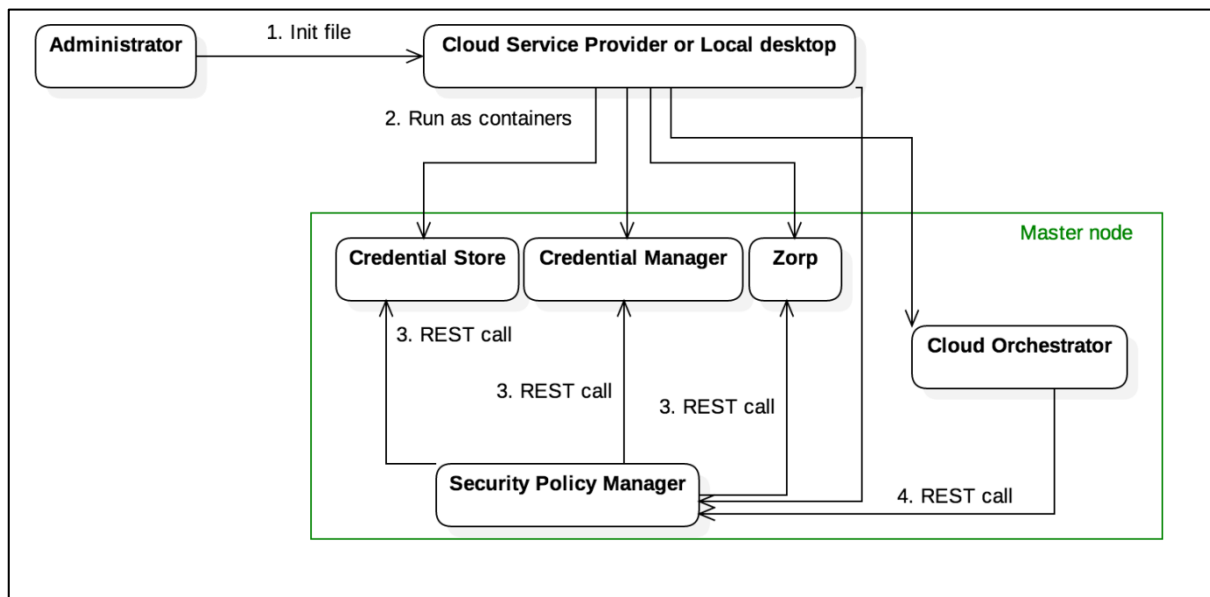


Figure 10 Security components initialization

1. Administrator generates an *init* file and runs it into the cloud service provider (CSP) or a desktop. This *init* file describes all core components of the infrastructure, including the security components.
2. The master node is launched with a list of components:
 - Security Policy Manager, i.e. SPM, as a docker container;
 - Credential Store, i.e. CS, as a docker container;
 - Credential Manager (i.e. CM, as a docker container);
 - Zorp as a docker container,
 - Crypto Engine, i.e. CE, as a library.

D7.4 Security policy formats specification

3. SPM executes
 - REST call to Credential Store to insert the cloud user credential in this secure storage;
 - REST call to Credential Manager to insert user account, that will be used to authenticate user later, into this secure storage;
 - Validates security policies in incoming Application Description Templates and signals other components if an application-level firewall container should be deployed along with the application;
 - Creates current Zorp configuration from exposed ports and network security policies and saves them to the distributed key-value store;
 - Signals Zorp instances to re-load their configuration.
4. Prior to spawning a new worker node, the Cloud Orchestrator executes REST calls to SPM in order to
 - Request a token SSL setting on the new worker node;
 - Retrieve ports required to be open on the new worker node;
 - Retrieve the cloud user credential to be able to send requests to cloud service provider.

According to the activity flow at launch time of the master node, it can be seen that there exist dependencies among components in the master node.

Table 17 Dependencies among components in master node

#	Dependency	Explanation
1	Security Policy Manager depends on Credential Manager	CM service needs to be ready before SPM execute REST call to add a user account
2	Security Policy Manager depends on Credential Store	CS service needs to be ready before SPM execute REST call to add cloud user credential
3	Cloud Orchestrator depends on Security Policy Manager	SPM needs to be ready before CO notifies the SPM that a new worker node is to be provisioned and request SPM to return a token for setting up Zorp SSL in newly created worker node

Such dependencies can be complied with by utilizing the start-up order configuration in docker compose [4]. The following example demonstrates how to configure it in the init file that may change later.

Table 18 Dependency between Security Policy Manager and Credential Manager

<pre> credman: image: my_docker_registry/credman container_name: credman expose: - 5001 spm depends_on: - credman image: my_docker_registry/spm container_name: spm expose: - 5003 command: > </pre>
--

D7.4 Security policy formats specification

```
/bin/bash -c "
    ./wait_for_it.sh credman: 5001;
    curl credman: 5001/v1.0/adduser -X POST --data '{"username":
"user 01", "password": "123", "email": "user 01@email.com"}'
    python ./my_script.py
"
```

Assuming that we have a bash script `wait_for_it.sh` that checks if a REST service is available, and REST API `credman: 5001/v1.0/adduser` that adds a user into the Credential Manager. As shown in the example, SPM depends on the Credential Manager component (CredMan). In addition, when SPM is started, it continuously checks if the service from CredMan on port 5001 is ready before calling REST API to add a new user into the Credential Manager.

5.4.2 Through the use of command line

While some infrastructure security features need to be initialized using the *init* file, others can be configured later by the administrator through command line using SSH connection. Furthermore, as soon as the administrator needs to change some security features, he can do it through the command line. For instance:

- Update cloud user credential;
- Update SSL certificate of the master node for MiCADO;
- Update the credentials of MiCADO master node users.

5.4.3 Extension

The infrastructure may be extended to support the administrator to configure security features by defining policies (through the *init* file) or updating policies (using command line). Has there has been no decision as of now on how to implement the description of such policies, we describe them in the format of tables. Each table contains a list of parameters for each defined policy along with its type and meaning.

1. Password policy

This policy defines how users should choose their passwords.

Table 19 Overview of password policy parameters

Parameter	Type	Meaning
PASSWD_MIN_LEN	Integer	Minimum length of password
PASSWD_MAX_LEN	Integer	Maximum length of password
UPPERCASE	Boolean	Indicating if the password must contain at least one uppercase letter or not
LOWERCASE	Boolean	Indicating if the password must contain at least one lowercase letter or not
NUMBER	Boolean	Indicating if the password must contain at least one number or not
SPECIAL_CHAR	Boolean	Indicating if the password must contain at least one special character or not
SPECIAL_CHAR_LIST	List	List of allowed special characters
RESET_PWD_AFTER	Integer	The number of days before the user is asked to be change their password

D7.4 Security policy formats specification

2. User account lock policy

This policy defines how the infrastructure deals with failed log in attempts (e.g. locking out accounts).

Table 20 User account lock policy parameters

Parameter	Type	Meaning
LOCK_DURATION	Integer	How long (in minutes) an account should be locked?
MAX_FAILS	Integer	Maximum number of allowed failed attempts to log in before the account is locked
SEND_MAIL	Boolean	Indicating if the system shall send email to the user whose account is locked or not.

3. Log policy

This policy defines how the system logs its error and notifies the administrator.

Table 21 Log policy parameters

Parameter	Type	Meaning
LOG_LEVEL	String	Indicating the lowest level that should be logged. For instance, LOG_LEVEL = INFO means that logging all information at levels INFO, WARNING, ERROR, CRITICAL but the level DEBUG should be logged
MAIL_NOTIFICATION	Integer	Indicating if the system sends notification email to the admin whenever errors happen
MAIL_LOG_LEVEL	String	Indicating the lowest level that a notification email should be sent to the administrator. For instance, LOG_LEVEL = ERROR means that logging all information at levels ERROR, CRITICAL should be notified by email to the admin.
MAIL_SERVER	String	Mail server
MAIL_SERVER_PORT	Integer	Mail server port
SENDER_MAIL	String	Mail address
SENDER_MAIL_PASSWD	String	Mail password
RECEIVER_MAIL	String	Mail address

4. Sensitive information access policy

This policy defines which components can access sensitive information stored in the master node.

Table 22 Sensitive information access policy parameters

Parameter	Type	Meaning
SECRET_NAME	String	Name of sensitive information. This name should match the name stored in the Credential Store component
COMPONENT_LIST	String	Name of components, i.e. docker containers, in the

D7.4 Security policy formats specification

		master node that are allowed to access the sensitive information
--	--	--

5. Database configuration

This configuration defines the location to store the credential database.

Table 23 Database confirmation parameters

Parameter	Type	Meaning
DB_PATH	String	Path and filename of the database

6. User reset password policy

This policy defines how the infrastructure processes the reset password request.

Table 24 User reset password policy parameters

Parameter	Type	Meaning
RESET_BY_LINK_OR_TEMP_PASSWD	Integer	Indicating if sending a temporary password or a reset-password-link to the user
VALID_DURATION	Integer	How long (in minutes) a temporary password or a reset-password-link shall be valid? If time is over and user does not change to their new password, the password keeps unchanged.

6 Summary and Conclusions

In this document, we identified the necessary security features for the MiCADO *infrastructure*. To this end, we identified features supported for applications as well as features that are directly associated with the infrastructure. With application security features, users can customize their parameters to make them suitable for their applications. However, only the administrator can configure the security features of the infrastructure.

Users can configure security features of an application by creating valid policies in an Application Description Template based on the TOSCA specification language. Each generated Application Description Template file will be given as input to the Submitter who then passes it to the Security Policy Manager (SPM) component. Upon reception, SPM sends a request to the relevant security components to deploy each security feature. This request can be processed automatically without the involvement of a user. Instead, the user just needs to worry about customizing policies that are defined in the Application Description Template file. Users do not need to know about how these features would be executed (e.g. by which internal components etc.).

Meanwhile, security components and infrastructure security features are presented in the *init* file. The security components are launched right after the master node is started. After that, security features are executed. This requires control on start-up order of the security components and their seamless interactions in the infrastructure.

In both cases, in order to apply application and infrastructure security features, it requires communication among different components in the infrastructures. In this document, we pointed out how these components would communicate to each other and how the configuration information will be processed from the user or admin to the internal component of the infrastructure that needs to be deployed. Although this might be changed in the future to adapt the actual implementation, this is a detailed overview on how the system works in order to provide basic security features.

7 References

- [1] D7.2 - MiCADO security architecture specification
- [2] <https://github.com/COLAProject/COLAREpo/blob/master/templates/dataavenue.yaml>
last accessed on 3 May, 2018
- [3] <https://docs.docker.com/engine/swarm/key-concepts/>, last accessed on 5 Feb, 2018
- [4] <https://docs.docker.com/compose/startup-order/>, last accessed on 19 April, 2018
- [5] <http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/TOSCA-Simple-Profile-YAML-v1.2.pdf>, last accessed on 21 Feb, 2018
- [6] https://github.com/COLAProject/COLAREpo/blob/master/examples/secret_example.yaml, last accessed on 10 May, 2018
- D7.3 - Design of application level security classification formats and principles
- [7] <https://github.com/openstack/tosca-parser>, last accessed on 28 June, 2018
- [8] D6.2 - Prototype and documentation of the monitoring service
- [9] D5.4 - First Set of Templates and Services of Use Cases
- [10] https://github.com/micado-scale/tosca/blob/master/policy/security/network/firewall_configuration.yaml, last
accessed on 28 June, 2018
- [11] Dierks, Tim, and Eric Rescorla. *The transport layer security (TLS) protocol version 1.2*. No. RFC 5246. 2008.
- [12] YAML Web Page. <http://yaml.org/>
- [13] Ansible Web Page. <https://www.ansible.com/>