This part of the thesis will present requirements analysis for a microservices-based web application that relies on content trust to verify the behavior of its different components. The discussion will be split into two parts:

1. Microservices and micro frontends
2. Content trust between microservices

The reason for such division is that, microservices and micro frontends are architecture. Hence they have their own set of rules and requirements when it comes to building an application based on this new architecture.

On the other hand, content trust is one of the security concerns, and with just like any application, security concerns will be addressed throughout the development of the proposed Blog. Yet the focus here is on the trust among different microservices, especially those coming from different sources/developers. Thus, this chapter will allocate a subsection in the requirements analysis to discuss the requirements analysis of the content trust of the application.

**Content trust between microservices**

This section will discuss the requirements for a content trust system that will be implemented to help microservices verify the behavior of other microservices. The following pages will try to give more details about what it should be included in content trust system.

Requirements for the content trust can be divided into different categories:

1. Microservices requirements
2. Requirements

The reason for such division is that, developers will need to implement and add extra modules of code to allow their developed microservices to handle content trust. Furthermore, microservices themselves need to have clear rules about how to communicate with other microservices especially ones coming from different developers. This communication must allow microservices to make a judgment of whether to trust the other microservices or not.

On the other sides, there are no requirements for the user, the one who is going to use the overall application. Because, content trust is something to be handled between microservices, or a protocol and set of rules to be read and understood by the microservices themselves. And so the user has no role in how the content trust should be and what it should include.

For the first division which is microservices requirements, it is what each microservice expect from another microservice.

On the other side, developers requirements refer to what each developer expect when writing a microservice that is going to work with other microservices. And what developers of microservices are supposed to add to their developed product to satisfy the content trust requirements of other developers.

**Microservices requirements**

A system based on microservices architecture consists of different small pieces of code. Each small piece is an application that can be deployed independently. It can also be updated and modified while no need for any modifications of the other small apps because such an update took place for one or more microservices. Such architecture, in theory, makes the system loosely coupled. Thus different system parts and components are easy to change, update, modify or even replace. As long as the interface of the new introduced microservices respects the old interface or keeps using the same communications protocols, then the system will continue to function.

Just like any other system, applications based on the microservices architecture are vulnerable to security threats. This vulnerability comes from the traditional security challenges that face any other system, but also from the nature of a microservices-based application.

Content trust has more dynamic nature than other types of trust such as identity trust. While when performing an identity check the outcome could be one of two. Either the identity has been proven, or the identity failed to prove itself. But with content trust several characteristics, measures and aspects should be taken into account to come to a decision of trust or distrust.

Those measures differ in how hard it is to obtain the needed information. It all depends on the kind of information is being retrieved. In order for microservices to be able to check the content trust of one another each microservice should be able check the following:

* The identity of a microservice
* The type of service a microservice provide.
* How sensitive the service provided by the microservice.
* How many other microservices interact and use the services of a certain microservice
* The evaluation of content trust of a certain microservice by other microservices
* Age of the microservice in concern in relation to other microservices
* The relation of the microservice in concern with other trusted microservices
* The interface and the nature of the connection with the microservice

The identity of a microservice

When an application starts using third-party microservices then there should be a need for verifying the microservices and also verifying their behavior.

Security concerns should be addressed for both cases. Each microservice should be able to verify that the other microservice that’s trying to connect with it is what it claims it is. Then once this established each microservice should be able to verify the behavior of other one.

The reason for this verification is because microservices are written in a way makes them easily deployed and modified. The whole idea of the microservices architecture is built on creating a very flexible system. Such system will continue to run when one or more of its microservices are not functioning. It will also continue to run when developers decide to make changes. Such changes could be that one or more microservices need update, some alternations or even a complete replacement.

Here at this point, when introducing new microservices to the system, the older microservices should be able to have a good security system. Such system will allow older microservices to check the identity as well as the behavior of the newly added microservices.

This mentioned scenario is the normal case. It is ok and even expected that developers of a system will make changes once the system is deployed. It is even an expected behavior, especially in the testing phases when developers are running beta versions of their newly developed application. In this stage developers could continue to make changes. Hence microservices are not supposed to exchange data blindly with other microservices.

The other scenario that could take place is when developers of the system decide to use one or more microservices that’s not designed by them. A third-party microservice. It will be ok if developers can verify the microservice themselves. But in case the system is composed of hundreds of microservices and developers used a handful of third-party microservices. In such case checking each microservice individually is not possible. Especially if some microservices depend on libraries developed by other parties. Hence the need for microservices to have a way to check and verify with other microservices before communicating or exchanging data.

The fear of hacking into the system is another situation that comes also from the nature of microservices. Such hacking could happen to replace one microservice with another one. The newly added microservice could successfully have a similar interface, allowing the other microservices to exchange data with it. In such case if unwanted party get to access and replace one or more microservices, they could also be able to access sensitive information exchanged with the new implanted microservices. Where those added microservices could send any information exchanged with them outside the application and thus putting the security of the clients at risk.

Verifications of other microservices

A system must be in place to help microservices verify each other and make sure at first that each microservice is what it claims. Verification is not an essential step for content trust but if verification was available. Then before proceeding to the contextual check, each microservice will have the opportunity to declare itself to others.

If a verification mechanism is implemented as a part of the content trust system, then the evaluation process will be handled in two steps. The system at first must allow for microservices to verify themselves to other microservices. If the verification process was successful the microservices move on to the next step which is the contextual check. In case the verification step failed then the involved microservices can’t move further in the process and any planned communication will not take place. Moreover such scenario could actually lead to further steps. The system could include a notification mechanism. Those notification will signal the ID of the involved microservices and which one failed in authorizing itself. Hence developers can take notes of what happened and make sure that all the microservices of the system are secure.

Microservices can always proceed to the content trust check without going through any identity verifications. But having an identity verification system implemented will help to make the system more secure. Hence a discussion of the different possible verification methods is presented.

Verifications of microservices can be implemented in different ways, the following methods will be discussed:

* One-to-one verification
* Centralized verification

One-to-one verification

When one-to-one verification takes place, each microservice will have the opportunity to exchange identity information with another microservice. This exchange of identity information should happen only once. The very first time two microservices are trying to connect with each other.

When a microservice is sending a request to another microservice, if both microservices never connected before then a protocol of identification should be followed. Such protocol could allow each microservice to check the identity of the other microservice and make sure that it is exchanging data with an authorized microservice.

When developers are implementing different microservice, a discovery record could be developed to keep track of the added microservices to the system. This way each microservice can search this list and know what microservices are available to connect with. Inside this index of microservices, each microservice can have its identifying information. Hence when two microservices are about to connect for the first time. Each one will send a copy of its public identity information to the other. The microservice on the receiving side, will check the received information against the ones available in the record. Information in the record is put by the developers themselves and is secured against hacking or any attacks.

Developers could also store the hash of identity data for each microservice in the record. Once two microservices wants to connect and exchange data. They will send each other their identity information. Each microservice will receive identity information and then calculate the hash for the received data. Once the hash is calculated it can be compared against the hash that was generated by the developers. If the hash values match then the microservice is what it claims to be. Because each microservice will not know how the hash was calculated. If someone was trying to put unauthorized microservice in the system, they will not be able have a microservice with identity information that will result in the same hash data as the ones available in the index of microservices.

Of course such approach doesn’t guarantee security of the system in all cases. And extra attention must be given to the index of microservices to keep safe from any attacks.

Centralized verification

Such approach can take many forms. One way of doing a centralized verification is by having a central microservice responsible for the verifications of different microservices. This approach is simple, the authentication microservice will have a table that contains certain secure information about each microservice. Once a microservice wants to connect with another microservice, it should first contact the authorization microservice.

The idea is similar to a log in process. Each microservice will send its secure data to the central microservice. Once received by the central microservice, those data will be checked. The authorization microservice will check this data against the data that it has already stored b the developer in its database. If there’s a match then, the requesting microservice will be informed. The next step would be that the requesting microservice will inform the central microservice of which microservice it wants to connect with. Once this information is received, the central microservice will send a token for both microservices that want to connect with each other.

Once the token is received both microservices can exchange this token and start their communication. This process is only needed the first time any two microservices want to connect together. Next time each microservice can start exchanging information directly with the other microservice.

As discussed above, verification can be skipped before content trust check is performed, but having verification of identity as a first step will help microservices to communicate with each other after establishing a basic security step.

This verification could actually play a role in the content trust and the decision of trusting the behavior of one microservice or not. In this case if a verification system is implemented and the verification step is passed successfully. Then an extra value could be added to the microservices which successfully authorize themselves to other microservices. Of course such consideration depends on the final system that’s implemented for the content trust.

**Literature review**

This part of the thesis tries to present a literature review of trust and the different mechanisms used to help secure the behavior of a software. The interaction between the user and the application as well as the interaction among different applications.

First, a definition of trust is illustrated, to give the reader a broader understanding of trust and its use in the literature. The next step is presenting the different techniques of trust as used by researchers and software developers.

Definition of trust.

The word trust has been a subject of many studies, many researchers tried to form a definition of trust and what it means. The reason for this is because trust plays an important part of people’s life and is involved in many scientific fields such as philosophy, psychology, economy and recently in computer science. In his famous PhD thesis Marsh [9] mentions that many efforts have been spent trying to discuss trust and putting a definition to it, especially in the second half of the last century. His research was an attempt to create a model that can offer a mathematical way to measure trust.

Dr. Harrison et al [6] concludes that there’s no one agreed upon definition of trust, “little consensus has formed on what trust means”. In his research, he agrees with Marsh [9] that many discussions have been written about trust. Furthermore he adds that different definition of trust are used in the literature,” significantly diverse definitions of trust continue to be used in the interdisciplinary research literature”. In his attempts to giving a definition for the word trust, Dr. Harrison et al [6] first tries to show the importance of trust. He mentions that “trust leads to cooperation”. He first, presents what some other researches have provided as a definition or an understanding of trust. On the other hand On the other hand Singh et al [18] try to give a definition or an explanation of how trust can be evaluated. Their idea is that the trust between two parties is a variable with many dependencies. They say: “The trust of an entity with other entity is not a fixed value but can change dynamically depending on the behavior of the entity and context in the environment”

Then Dr. Harrison et al [6] distinguish between six types of trust “we define six related types of trust”. While Singh et al [18] make a distinction between two types of trust, those are execution trust and code trust. They explain them as “Execution trust exists from subject’s side to service provider’s side that service provider will correctly and faithfully allocate resources for the efficient execution of job with respect to established policies”. While according to Singh et al [18] code trust “exists from service provider’s side to subject’s side that subject will generate a legitimate request consisting of virus free code and will not produce malicious results and does not temper other results/information/code”

Moreover Singh et al [18] give other distinctions for the trust, they explain this as trust could have other types. This extended distinctions of trust types is composed of seven types of trust. Namely “direct trust, indirect trust, full trust, partial trust, recommended trust, authentication trust and finally privacy trust”.

The first type of trust defined by Dr. Harrison et al [6] is the Trusting Intention. This type of trust means that one is able to depend on others. Dr. Harrison et al [6] argue that this type of trust is a “situation-specific”. On the contrary to this definition Gabarro, J. J. [10] thinks that this type of trust is not a situation specific. Dr. Harrison et al [6] present the difference in such understanding as “This makes the other person the object of trust, rather than the person in one situation”

The second type of trust is the “trusting behavior” according to Dr. Harrison et al [6]. The definition for trusting behavior is also given by Lewis & Weigert [11] “is the extent to which one person voluntarily depends on another person in a specific situation with a feeling of relative security, even though negative consequences are possible.“ Dr. Harrison et al [6] go further and tries to decompose trusting behavior into different subconstruct forms. Namely are: “cooperation, information sharing, informal agreements, decreasing controls, accepting influence, granting autonomy, and transacting business. “ Anna Macko & Marcin Malawski [12] studied the trusting behavior in their work, named “Belief in others’ trustworthiness and trusting behavior”. They show that many factors plays a role in the trusting behavior, and it is not just about “maximizing the profit”.

Dr. Harrison et al [6] puts “trusting Beliefs” as the third type of trust. The given explanation is “the extent to which one believes (and feels confident in believing) that the other person is trustworthy in the situation” other researches have also studied trusting beliefs, for example Dr. McKnighta et al [13] explain that trusting beliefs are “perceptions of the competence, benevolence, and integrity”. In their explanation they give an example of a vendor-consumer relationship. “Willingness to depend (that is, a decision to make oneself vulnerable to the vendor)”.

In [] a discussion about the importance of perceived information and its consequences. When low-quality information is provided but such information where not of a big importance then the consequences of such false information is relatively low. However, when the nature provided low-quality information has high importance then the negative consequences of trusting these information could be high.

Moving on to the fourth type of trust that was distinguished by Dr. Harrison et al [6], this type is called system trust. It is explained as “means the extent to which one believes that proper impersonal structures are in place to enable one to anticipate a successful future endeavor”. Robin Pennington et al [14] give an example of system trust, the use ecommerce system as such example. They concluded that system trust has an impact on the intentions of customers to decide whether to buy or not, “system trust plays an important role in the nomological network by directly affecting trust in vendors and indirectly affecting attitudes and intentions to purchase.”

Dr. Harrison et al [6] provide dispositional trust as the fifth type of trust, explaining “, if one believes that others are generally trustworthy (Belief-in-People), then one will have Trusting Beliefs (which in turn lead to Trusting Intention).” Dispositional trust is also noted by Sharon G. [15], in his research about situational uncertainty, he concludes that dispositional trust can foresee trusting phenomenon.

Lastly, the sixth type of trust according to Dr. Harrison et al [6] is the situational decision to trust. Dr. Harrison et al [6] explain it as “the extent to which one intends to depend on a non-specific other party in a given situation. “. Although it is recognized as a different type of trust, but it doesn’t exhibit much difference from the first type of trust which is Trusting Intention.

Dean Leith [8] in his paper about the concept of trust defines trust as “Trust as a ‘leap of faith’ or willingness to be vulnerable”. He argues that trust is a tool learnt at as earl age as infancy. Where people use this tool to approach uncertain situations “trust is learned in infancy and enables the individual to deal with the unknowable in the social context”. In his explanation for the term ‘leap of faith’ he presents this term as an important part of the trust where it “involves the trustor experiencing a lack of expertise in a particular area of their life and acknowledging that the expertise they require to address this lack is held by another individual or system. They either consciously“

However another definition of trust is also presented in [8] trust is seen as a “social capital” writer argues about the role trust plays for individuals in society and hence the role each individual plays in the society.

Lastly [8] also presents trust as a part in the “power-knowledge” theory where knowledge leads to power and trust is an important component to acquire knowledge.

As can be seen that there’s no one definition of trust in the literature and many researchers have come to conclude different means and concepts of trust. Some have given examples from the real world such as Robin Pennington et al [14] where he talks about trusting a system. The same concept of trust is agreed up by Dr. Harrison et al [6] where he gives an example of trust in a system of doing a purchase via the credit card. Where both the buyer and the seller trust the system. In case the system rejects the credit card of the buyer, both parties will not have less trust of the system but the seller will only be suspecting the buyer and most likely never the system.

Authentication and Authorization

Authentication and authorization are important for the trust. When a service is able to identify itself to other services, it helps to add points to the overall evaluation of the trust. Authorization will help to have the requestor gains access to the resources such as data. Singh et al [18] defines authorization as “deals with issues like who can access which resources/services under which conditions”. Hence once a microservice is authorized, it will be able to make requests to other microservices and exchange data with them.

Singh et al [18] try to give an explanation for an authorization system by describing it as a system that provides access rights. They explain “An authorization system can be defined as a system that grants specific type of access to specific requesters based on their authentication”

Ramalingam et al [19] describe authentication as “allows identity verification of any entity.” Moreover, the authentication of users as “the basic feature of protecting data from computer system intruders”. Wallace [20] agrees to this definition, he states an authentication protocol as “its purpose is to authenticate entities wishing to communicate securely. “

Conklin et al [21] go as far as describing authentication as a very important aspect of computer systems security, they mention “authentication in computer systems has been a cornerstone of computer security for decades” such statement shows without doubt the importance of authentication in computer system and applications and agree with the rest of the researchers about its importance and role. Wallace [20] continue to explain authentication as “Authentication is a simple function where one party presents a set of credentials to a system. If the credentials match a given set on the system, the system returns a value that represents authorization; otherwise it does not.” In the case of microservices, this attempts of authentication happen between microservices. Where one is trying to identify itself to the others. Similar to Ramalingam in [19], Wallace [20] continues to present the importance of authentication by stating that it is the very first step a requestor has to take before it is granted further access, “verifying the identity of an entity is the basis for all future rights and privileges granted to the entity”.

Furthermore, Conklin et al [21] try to give a simple example of what an authentication process could look like. Their example uses the ID of a user for achieving an identity verification. They say: “In the basic authentication process, the entity desiring authentication presents credentials, usually an account ID and some additional information, to prove that the request is coming from a legitimate owner of the ID.” Ramalingam et al in [19] shows the importance of authentication as it is the proceeding step before authorization “Authentication is critical for sending our data over the internet, as well as for ensuring that authorizing is done properly allowing access to systems and services.” Conklin et al [21] agree with Ramalingam et al [19] to the importance of verification where “Identifying a user is essential for the application of security in the form of permissions to various objects, processes and access to resources”

Merkow et al [22] in their book “Information Security: Principles and Practices” enclose the goals of security in three point: “All information security measures try to address at least one of three goals” those points are stated to be “Protect the confidentiality of data, preserve the integrity of data, promote the availability of data for authorized use”. On the other Ramalingam et al [19] state similar points as the task of authentication and authorization. They state that “Authentication and authorization is implemented to ensure confidentiality, integrity, availability, authenticity, and accountability.”

Leslie Wallace [20] indicates to the importance of identity check in computer system. He explains that having identity verification helps in making the system more secure against attacks. “Validating users before allowing them access is an easy way to catch an intruder from stepping into boundaries that they shouldn't cross”.

Ramalingam et al [19] continue to explain how authentication is achieved. Where they explain that “Authenticating users is carried out by the series of identification and verification stages. At the identification stage, access to a security system is defined, and the binding between an entity and an identifier is done during the verification step.”

From the definitions and explanations given by different researchers, it is clear that authentication is an important step in giving access rights to a requestor that’s trying to access one or more resources.

Conklin et al [21] try to provide a more practical view on authentication by presenting a simple mechanism which uses a combination of a username and a password. They state “The concept of a user id and password is a cost effective and efficient method of maintaining a shared secret between a user and a computer system”. Conklin et al [21] move on explaining that many computer systems use the simple known identification method. This method is composed of a username and a password, “Many modern systems have adopted a simple id/password method of achieving the goals associated with the identification and authentication function”. The same thing that Ramalingam et al [19] agree upon, they say “the old-fashioned technique which requires a username and password remains the prevailing measure of securing computers, email accounts, or online transactions.” They also explain that despite all the advances that took place in both hardware and software, the combination of username and password is still the most dominant way of verification in computer systems. Saying: “Even with the adoption of new technologies in hardware and software, password authentication is still not completely replaced by the existing alternative authentication methods.”

On the other hand Singh et al in [18] give more in depth definition of an authentication and authorization system. They define each entity in the process from the requestor to the requestee including the resources and the action to be taken upon these resources. They describe the requestor as “Subject is an entity that wants to access services/resources. It can be a user, a service or any other entity on behalf of user/service”. And then move on to describe the service by explaining that “Service is a piece of software that provides some functionality and can be accessed by Subjects or other Services”.

After Singh et al in [18] have described the requestor and the service they continue by giving a definition to the resource that’s being requested. They say: “Resource is an object that is accessed by Subjects. It can be a CPU, a storage device, software, data”

Another interesting definition given by Singh et al [18] is about the requirements given by each service in order to be accessed. This is called Service Policy, explained as “Service Policy refers to the set of rules/requirements associated with the Service. A Subject must conform to Service Policy in order to Access that Service”. Furthermore, the access that’s granted to reach the requested service is also explained. Where “Access is an operation that a Subject performs on Service/Resource. The access is provided based on conformance to Service Policy that is associated with that Service/Resource. Hence it can be clearly seen by Singh et al [18] definitions that access to the service is not granted unless the service policy of the service is respected. While Policy itself is also described by Singh et al [18] as “a set of rules/requirements”. Where this set of rules can be linked to the Subject, the Service or even the Domain according to Singh et al [18].

Reputation based trust

Reputation based trust started as a review made by users for others, one of the earliest examples of it was adopted by eBay. As [23] refers “Reputation systems are already being used in successful commercial online applications”. A similar idea is referred to in [28], it says: “Reputation-based trust systems were mainly used in electronic markets, as a way of assessing the participants”.

However, trust has been divided into two distinctions one as “strong and crisp” while the other as “soft and social” [24] the “soft and social” is concerned with reputation based trust. Where according to [24] “reputation-based trust relies on a “soft computational” approach“. In this case, trust is computed from two sources: First based on own experience, second based on experiences of others as referred to by [24]. Moreover, trust depends on other factors such as the time and the settings [26].

The same concept for computing trust is used in [25], it agrees that reputation-based trust is computed from two sources. It says: “Reputation information is divided into two categories: First-hand experiences are gained from the trustor monitoring the outcomes of actions it has engaged in itself, and are generally considered to be error-free within the limits of observed- ability. External experiences are gained from third-party recommenders based on their own first-hand experiences”.

On the other hand, in [26] uses the term “behavioral trust” instead of “reputation trust”. It is defined as “fulfilling the expectation of others”. And it is classified into two categories: Direct trust and indirect trust. Where direct trust means the experiences gain from own direct interaction. While indirect trust means the experiences of other’s interactions. It is obvious that [26] agrees with [25] where it uses first-hand experiences and external-experiences instead of direct trust and indirect trust. In both definition, the resulted trust is variable and never constant, where its value changes after each interaction. While in policy-based trust the resulting decision is either positive or negative [24]. Where such trust depends on well-defined measures such as certificates and is referred to as “strong security” [24].

[27] Also agrees with the mentioned studies, it states that “reputation serves as the basis for trust”. Hence giving an important value for the experiences of other entities in the system.

**Analysis**

This section provides an analysis of the content trust as well as microservices architecture in regards to the requirements that we presented in the first section of this chapter. Those requirements will be analyzed against the presented literature review in the second section of this chapter. The analysis will be discussed under two titles:

* Microservices
* Content trust

Besides going through the previously presented literature review, the following pages will also compare the presented requirements against some well-known implementations of microservices as well as content trust.

Content trust analysis

Different definitions of trust were presented by different researchers. Where some of the definitions intersect with one or more of the provided requirements of content trust. One definition for trust distinguish between direct trust and indirect trust. Direct trust is established after own direct interaction with other entities. While indirect trust information is gathered from other entities experiences with the entity in concern. This definition can be seen by in least one of the requirements of content trust. Microservices will have their own evaluation of trust once they interact with a certain microservice. And also will be depending on the evaluation of other microservices for the concerned microservice.

After each interaction the trust they already have about the other microservice could be affected positively or negatively. Moreover their new evaluation of trust could also play a role on how other microservices evaluate their trust with the concerned microservice. This is perceived from the indirect interaction, hence the indirect trust.

One of the widely cited study about trust [6] it is mentioned “trust leads to cooperation” this understanding is also exhibited in the requirements of content trust. The point of adopting content trust in microservices is to make collaborating microservices trust each other’s behavior and exchange data safely. In such scenario, having high evolution of content trust among microservices will lead to more exchanged data and cooperation.

Trusting the behavior of others was also mentioned by some researchers [6][11]. One definition is presented as “Belief in others’ trustworthiness” this definition is reflected in the evaluation of trust each microservice will have. Believe is reflected as a dynamic value that can grow or shrink depending on the how positive or negative each interaction is.

The relationship among services is present in more than one requirement. On the other hand, the relationships among collaborating entities was mentioned by different researchers. Such collaboration between concerned entities is mentioned in the sixth definitions of trust presented in [6][15]

The age of microservices is taken into account when deciding about the content trust of a microservice. It is mentioned as one of the requirements of the content trust system. However, the age doesn’t have much influence on any description provided for the trust or any of its contrasts or sub definitions. Yet the age is mentioned specifically in [content trust paper]. It is stated that the age of the content could play a role in helping the readers of the content on deciding whether the content is trustworthy or not.

The sensitivity of the service is also presented in the requirements of the content trust. In [] the importance of the provided information is discussed. In the requirements of content trust, the sensitivity of the service can be projected into the importance of information presented in []. In such case when there’s a trust among microservices that are exchanging sensitive information, but the information where not of high degree of integrity. Then as indicated in [] the consequences could be more serious, than if the exchanged information where of low importance.

The identity of a microservice has a weight in deciding of trusting the behavior of a microservice or not. In the requirements of microservices, each microservice should be able to authenticate itself to other microservices. Failing to authenticate itself, could result in having a decreased evolution of the service by one or more other services. In [21] a model for verifying the identity of a requestor is presented as a combination of User ID and a Password. Such combination is also used by other researchers []. In the case of microservices, proving each service what it claims to be is important as it establish a first level of trust. Such ground could be used to move on and try to establish other forms of trust. Having the identity of the service verified will help in increasing the evaluation of it by one or more services positively.

Regarding the implementation of content trust. The most common one is used by Docker and it is called Docker content trust. Docker is basically a container for processes. One can think of it as a virtual machine but much lightweight and faster to boot. This lightweight virtual machine is called a container and one host can have more than one container running at the same time and sharing the host resources. Docker containers are actually used widely to deploy services for applications built on the microservice architecture. Docker content trust is used to help in trusting the images of the containers released by software providers. According to the official documentation, the point of Docker content trust is to ensure the integrity of Docker images and also verify the identity of the publishers of the image. This explanation only satisfies a portion of the requirements of content trust presented in the first section of this chapter. But it doesn’t go any further, for example: users of Docker images can’t provide a evaluation of their experience after using a specific image. Hence other users can’t use such information in helping them to decide of whether it is reasonable to put one’s trust a specific image or not.