Agent-Based Evidence Collection in Cloud Computing

Philipp Ruf(✉), Thomas Rübsamen, and Christoph Reich

Cloud Research Lab, Furtwangen University of Applied Science,
Furtwangen, Germany
{philipp.ruf,thomas.ruebsamen,christoph.reich}@hs-furtwangen.de

Abstract. Nowadays there are many offerings of cloud services all over the world which have various security requirements depending on their business use. The compliance of these cloud services with the predefined security policies should be proven. In a cloud infrastructure this is not an easy job, because of its immense complexity. This paper proposes an architecture which uses software agents as its core components to collect evidence across the different layers of cloud infrastructures (Cloud Managment System, Hypervisor, VM, etc.) and builds a chain of evidence to prove compliance with predefined security policies.

Keywords: Cloud computing · Evidence · Persistence · Accountability · Audit

1 Introduction

This work addresses the problem of collecting, processing and persisting evidence from different sources inside a highly dynamic environment and its automation dependent on a policy describing the contract between a `costumer entity` and its `Cloud Service Provider (CSP)`. The automation aspect is addressed by an `Software Agent` (in following `agent`) and is based on the previous work "Supporting Cloud Accountability by Collecting Evidence Using Audit Agents" [1] by Prof. Dr. Christoph Reich and M. Sc. Thomas Rübsamen. For example, a `CSP` is collecting information about policy violations and storing them using the service of a second CSP. The potentially confidential captured evidence need to be persisted in a integrity-verifiable way and protected from unauthorized access. This project's expected contributions are additional control mechanisms highlighting the transparency desired by the cloud service costumers and accentuating the trust in `CSP`s and their contractual awareness.

The proposed agent-based architecture, which we describe in the following, collects evidence to allow the detection of policy violations and generates policy violation reports while protecting sensitive information and respecting costumer privacy at the same time. Thereby using an agent framework supporting strong and weak agent migration [2] was necessary for distributing and delegating tasks on demand adjusted to their different destination environments. The data to collect is depending on the assured policy contract between a cloud costumer and a cloud provider which, can be statutorily regulated, defined by the service provider or created out of user specific criteria. Through periodically audits, the implemented agents are able to provide the requested claims of evidence by persisting recognized policy violations.

While a Cloud Service itself potentially contains service interdependencies with external Service Providers, the sources of evidence to be covered by a trusted service are increasing, too. A `Chain of Accountability` can be formed implementing these centrally coordinated trusted (accounting) services along the supply chain. Pointing out the exact location of a occurred policy violation depicts how trust in a `CSP` is strengthened using services implementing evidence reveal- and notify mechanisms and therefore supporting accountability (e.g. members of the architecture to propose). In a multi-`CSP` scenario with service coherences inter-CSP collaboration is still a fundamental requirement. Currently, there is no standardized way for a cloud service costumer to check on his own whether or not he is affected by a policy violation occurred along the supply chain. The meanwhile established usage of `Web Objects` (like for example the `Amazon Simple Storage Service AS3`) extends the `Chain of Accountability` with dynamic interaction, which (usually) is transparent to the costumer. Potential evidence sources like these connections `on demand` and their potentially scalable content must be observed as of the time a active interaction with the service occurs. Therefore, the possibility of interacting with a `CSP`'s trusted service provides the transparency needed in a complex environment like the cloud.

This paper is structured as follows: in the first chapter of this paper we discuss related work (see Sect. 2). In Sect. 3, the evidence collection and persistence architecture including used technologies and agent coherences is described. Following that, the actual collection of evidence and the different collection agent types are explained in Sect. 4 followed by the persistence mechanism in Sect. 5. Following that, Sect. 6 describes the migration of agents in a scenario, where multiple cloud providers are involved. After that, an example of how service coherences are affecting policy violation evaluation, will be discussed in Sect. 7. We conclude this paper in Sect. 8 where our perception of further work is noted, too.

2 Related Work

Reich and Rübsamen empathized the need of policy violation audits and proposed how evidence collection has to be mapped to accountability in their previous work [1]. They are proposing an *Audit Agent System* which was the groundwork for the construction of the architecture presented in this paper. This work will not focus on the mentioned audit aspects but on the storing, presenting and processing of evidence.

The idea of using `Digital Evidence Bags (DEB)` [3] plays a key role as a solid evidence persistence structure in this paper. Based on the work by Turner, Schatz and Clark propose an extension for connecting evidence composing and referencing DEBs using a `Sealed Digital Evidence Bag` [4]. This mechanism is a possible extension to this architectures persistence mechanism.

The usage of software agents also were proposed in the context of `Security Audit as a Service (SAaaS)` [5]. A related presentation layer and distributed sources of evidence are discussed in this paper but specializing on security policies and the guardedness of a source located at the input layer. Also, a *'security business flow'* modeler generating the policies to observe is differencing `SAaaS` from approach proposed in this paper.

Of course there are many tools offering a agent-based solution for monitoring network devices by collecting and analyzing a wide range of current system properties. Industry standards like `Nagios` [6] also providing an agentless monitoring solution which is a less capable but goes easy on resources. Also there are Software as a Service (SaaS) monitoring solutions like `New Relic` [7] which providing agents collecting data dependent on different scopes and devices. Besides the traditional monitoring of (network - e.g. cloud) resources it supports real-time analytics and a performance monitoring, which can be integrated into the application development process. Therefore, it is not surprising to be confronted with this system using a Platform as a Service (PaaS) like `Cloud Control` [8] or `AppFog` [9] simplifying the monitoring of scalable applications. Transmitting the current values to the cloud brings different advantages like rapid data analyzing using resources on demand. The architecture proposed in this paper also supports monitoring functions but differs by extending this aspect in focusing on active intervention like interacting with third party tools and their provided `APIs`.

3 Architecture Overview

The focus of this section is the brief introduction of this works architecture design including its components and used technologies. To understand the flow and properties of the proposed architecture, we have to take a closer look at the `Java Agent DEvelopement framework (JADE)` [10] which is the technological foundation of our work (Fig. 1).

`JADE` complies with the `Foundation for Intelligent Physical Agents` (FIPA) specifications [11,12], which define the internal behavior on action selection and execution as well as external agent interaction. The external agent interaction refers to the interaction context and the message creation, which draws on the `FIPA ACL` (Agent Communication Language). Other specified parts of JADE are system and platform services, which can be used for agent service registration or agent migration [13].

Using this powerful framework makes creating different agent infrastructures quite simple. Every JADE instance is denoted as a `Container` while multiple `Containers` are denoted as a `Platform`. Inside a `Platform` exactly one `Main Container` exists beside the different agent implementations, which itself contain agents for infrastructural provisioning. New agents, for example, can be added transparently as needed and contacted after their registration with a platform's `Main Container`. Using the recommended design guide [14],

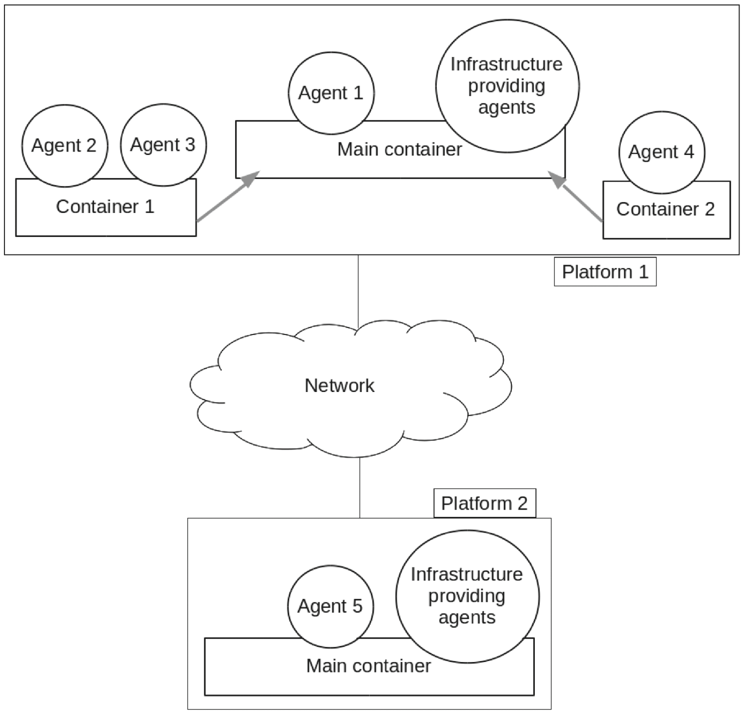

Fig. 1. JADE platforms and containers

the distributed agents are geared to each other and can be modularly extended. Also, the possibility of using `JADE` in `Public Key Infrastructure` mode is given by creating a configuration file containing the path to a `Key-Store`, key access password(s) and starting an agent with the configuration file as a parameter. Not only exchanged messages but also code transmission (used for agent migration - see Sect. 6) is encrypted if both participants are using the `JADE Public Key Infrastructure (PKI)` module/Add-On.

Being acquainted with the used agent framework lightens the contact with and comprehension of this architecture. A high level overview of the architecture is depicted in Fig. 2, revealing how distributed agents are communicating within it and in which way the different architectural components are interacting. All parts of this architecture are positioned inside one `JADE Platform` containing an evidence interpretation as well as a persistence agent which both interact with the distributed collection agents. Note, that additional services, which are provided out-of-the box by `JADE` (such as the centralized service registry and multi-platform interaction), are not pictured in Fig. 2.

To provide a chain of evidence, every trusted service of the supply chain contains a controlled agent which is responsible for evidence collection. To determine a policy violation, a variety of sources like the cloud management system, network packet data flows and storage units are browsed for conspicuous patterns. Also, external programs can be triggered to analyze their output.

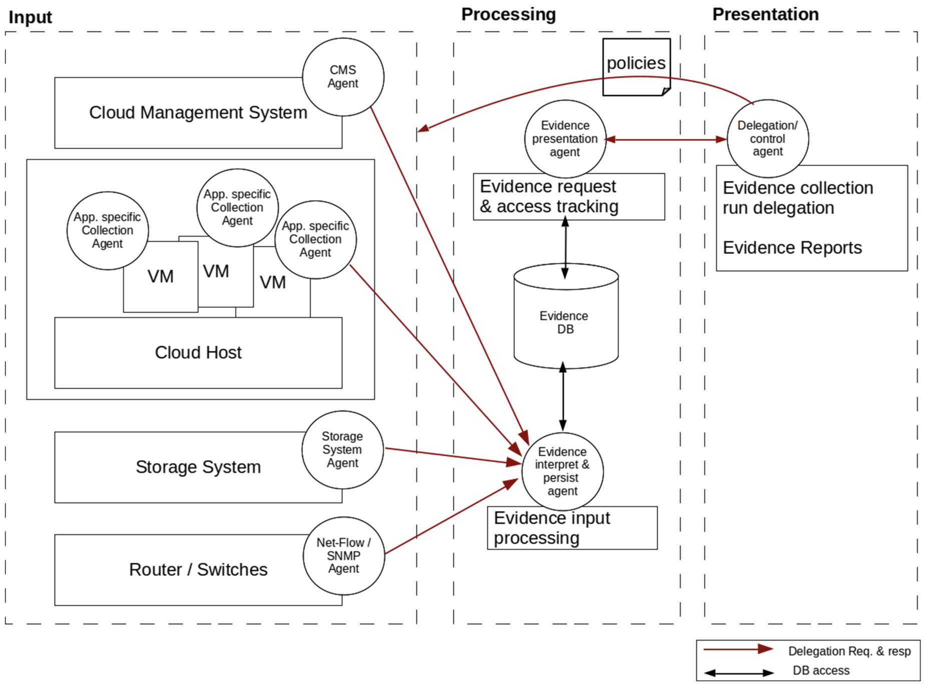
Input
Processing
Presentation
CMS Agent
Cloud Management System
App. specific Collection Agent
VM
Cloud Host
Storage System
Storage System Agent
Router / Switches
Net-Flow / SNMP Agent
policies
Evidence presentation agent
Evidence request & access tracking
Evidence DB
Evidence interpret & persist agent
Evidence input processing
Delegation/ control agent
Evidence collection run delegation
Evidence Reports
Delegation Req. & resp
DB access

Fig. 2. High level architecture

To ensure data and therefore evidence integrity, every evidence collection run is persisted using a `Digital Evidence Bag` [3] which has been adopted for this approach (see Sect. 5). The DEB can contain raw data and diverse meta-data depending on the costumer contract or the kind of occurred policy violation, respectively. This architecture provides an interface delegating and coordinating tasks by interacting with the distributed `Evidence Collection Agents`.

The `Input Layer` contains different agents responsible for collection of evidence in their different scopes and locations. Currently covered evidence sources and their techniques are specified in Sect. 4. Each detected violation is noticed and the whole evidence including its meta- and additional control data is sent to the processing side. To guarantee information integrity, evidence is provided with a signature, which must be consistent during the evidence report process. This means, evidence integrity is ensured from the violation detection until the conversion from raw data to the output message has happened. At this point, a participating actor or an external system could be notified automatically. This automatic, transparency-strengthening process could therefore strengthen the trust in a `CSP` providing composite services [15].

The `Processing Layer` is responsible for the integrity observance and the evidence request access tracking which is expressed inside the `DEBs Tag Continuity Blocks (TCB)`, implemented as a H2 Database [16]. The `Evidence`

`Interpret And Persist Agent` is listening for successfully finalized or failed evidence collection records created by the Input Layer which triggers the persisting of new insights. Also, the `Evidence DB` is communicating with the `Evidence Presentation Agent`, which interacts directly with the `Delegation/Control Agent`. Its only purpose is the expressive presentation of requested evidence in due consideration of the requesting actor. It is also conceivable to provide a costumers exclusive evidence collection system. In this scenario, the compliance of policies (from CSP side) is possible using a separate `Cloud Service Provider Agent` with the required access rights, respectively the interaction with the costumers exclusive `Delegation Control Agent`.

The `Presentation Layer` is the only point of direct contact with a human actor (e.g., a customer or a trusted third-party auditor) inside this architecture. The user interaction handling and request transformation is handed over to the `Delegation/Control Agent`. Besides requesting meta data about currently active `Evidence Collection Agents`, a costumer entity is able to check the current status of the contractual compliance with its CSP. Also the explicit delegation of a evidence collection run is possible due communicating with this agent. Conveniently, the opportunity of adapting the JADE library to JSP based systems is given. Therefore, the orchestration of agent actions (which of course must follow agreed-upon policies) could be added for instance to a costumers private web interface.

Protective goals inside this forensic mechanism are integrity and confidentiality of collected data which have to be guaranteed until the evidence collection has finished and was persisted. On the other hand, collected data should only be requested by authorized auditors or other authorized entities such as cloud regulators.

4 Evidence Collection

This section is an introduction to the different evidence collecting agents, their evidence sources and the detection of policy violations. As shown in Fig. 2 the `Input Layer` potentially contains several distributed evidence sources located at Infrastructure as a Service (IaaS) -level. Besides the examination of log files [1] inside different systems, APIs and external applications are sources of evidence, which can be used to determine policy violations by different patterns. The evidence collection is performed either by using a so called `OneShotBehaviour` for a particular evidence collection initiation or by collecting evidence periodically/continuously. Currently the evidence sources are covered by the following `Evidence Collection Agents`:

– `CMS Agent:`
  This agent is able to interact directly with the central component of any cloud infrastructure. To detect policy violations, `APIs` provided by the `Cloud Management System` are used to gather needed information. In case of OpenNebula [17] the process of evidence collection could be the request for current storage, network or virtualization orchestration and of course the analysis of

log files, where events originating from cloud operations are recorded. For example, there is this internal project working on business secrets, which is placed on a separate hypervisor. Because of its critical data, this system would claim besides other transparency increasing measures the delegation of a `CMS Agent`. This `agent` would be responsible for gathering lifecycle information, tracking of occurred snapshots (which are relevant considering the aspect of needed confidentiality) and workload information of every (reachable) node.
– `Application Specific Agent:`
  These agent types collect a specific kind of evidence defined by the policy. For example, a policy could look like "It is not allowed to store email addresses inside a VM". To detect evidence of non-compliance, patterns matching email addresses need to be searched and recorded. Therefore, the agent is triggering an external program searching the VMs hard disk for the given pattern. This can be done using the `Cornell Spider` tool [18], which generates a log file containing all file paths that possibly compromise the given pattern including additional meta data. Of course the occurrence of false positives cannot be excluded automatically from the output.
– `Storage System Agent:`
  This agent is communicating directly with various `Storage Management APIs`. Besides performance monitoring this agent is also able to determine the exact location (e.g. datacenter) of a service. This feature can be used to verify the awareness of policies requiring a geographically aspect.
– `Net-Flow Agent:`
  This agent's task is the investigation of different network-enabled devices inside a CSPs network. Some policies will prohibit the network communication with certain addresses and/or address-ranges for a specific network device. By analyzing NetFlow logs, the policy violating communication can be tracked and used as evidence (e.g., communication endpoints, time and duration).
  In Sect. 6 there is a description of how to use this agent type along different CSPs in case of using their `XaaS` as supply chain.

After the evidence collection run has finished, the executing `Evidence Collection Agent` generating a evidence record as basis for the corresponding `DEB`. In some cases the evidence is a complete file which must be sent to the `Processing Layer` for purposes of conservation of evidence. Working large log files containing evidence can become a performance problem because each file containing a violation must be transmitted to the processing layer. In that case the evidence file must be transmitted inside a signed `Blob` and persisted at the processing layer (best encrypted, too) guaranteeing tamper evident properties.

5 Evidence Persistence

This section illustrates the different data structures (`Tables`) of the `Evidence Database` and how the different attributes are mapped to and reflect a `Digital`

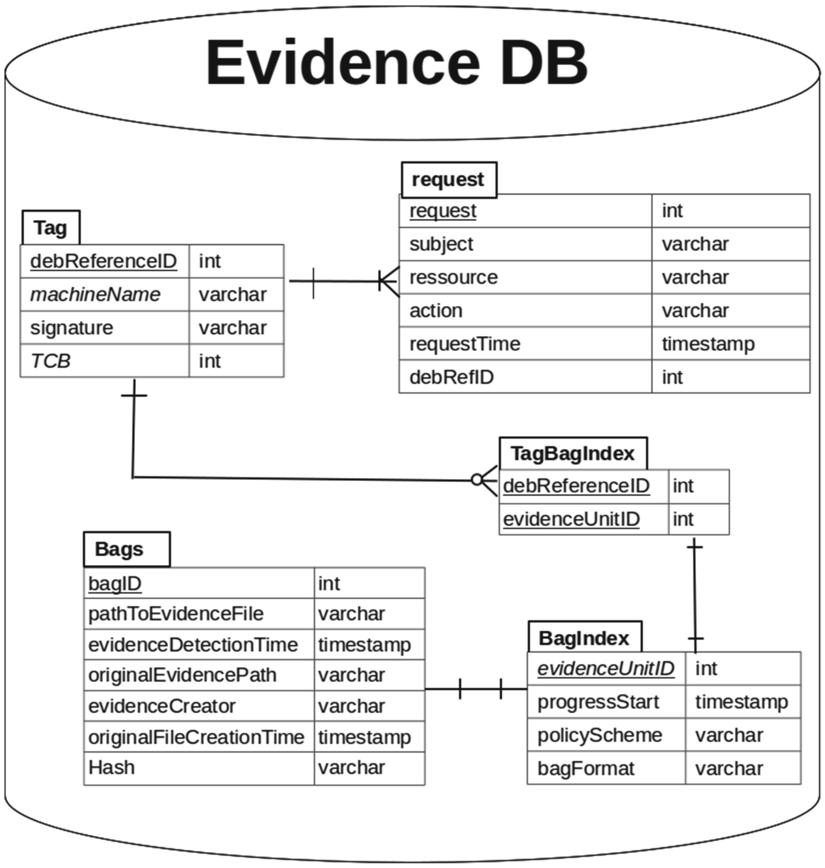

Fig. 3. Evidence database

`Evidence Bag (DEB)`. Like mentioned before, the persistence of evidence is triggered at the processing side as soon as the `Evidence Interpret and Persist Agent` is receiving a record from an `Evidence Collection Agent` registered with its platform. After verifying the dataset's integrity and adding the current `Request` record, every evidence request is processed by automatically forming the responding `Tag` row and its underlying layers. Therefore, the `Evidence Interpret and Persist Agent` is connected to a H2 `Evidence Database` (Fig. 3) using JDBC.

A `DEB` contains a `Tag` which has a `1:N` relation to an `Index`, which in turn is related `1:1` to a `Bag`.

– The `Tag Table` containing information about each evidence collection run in all because of its relation to the particular `Bag` and their overlying `Index Tables`. At this layer, meta data about the evidence record, information about the delegating agent and a reference to the last accessing subject of this evidence collection run are stored.

– To be aware of actions performed on a specific `Tag` or `Tag`-underlying layer the `TCB` entry references a row inside the `Request Table` to detect every single corresponding evidence access.
– The `1:N` relation between the `Tag` and the `Index` is established using an intermediate table.
– Also, containing a signature of every underlying relation altogether, the `Tag` table is a robust core element of this evidence persistence mechanism. Inside the evidence database the signatures are stored as `Base64 varchar`(String) and being restored as byte array for data integrity verification using the `Java.security API`.
– Besides the `Progress start` (at the `Evidence Collection Agents` side) and the `Policy scheme`, the `Index` table contains a `bagFormat` attribute to categorize the occurred evidence. As mentioned before every evidence collection run that scored no policy violation must be persisted, too. This can be done at this point. The `bagFormat` is a first indicator of the significance of persisted evidence. It can be *'structured text', 'raw binary data', 'archive', 'no policy violation'* or any other suitable categorization.
– By referencing the `Bag` table by its corresponding `evidenceUnitID`, a requesting subject is able to reach a stored policy violation. If the evidence is associated with a file, the file is persisted including its signature at the processing layers storage but is not stored inside the database because of performance loss inside the signing and verifying mechanism. However the path to this evidence file is stored inside this `Bag` table besides the `evidenceDetectionTime`, original evidence meta data and an additional Hash. Therefore, using the `JAVA Security API` every Tag is signed with the `Evidence Interpret and Persist Agent`'s DSA key, while the actual bag holds the `Evidence Collection Agent`'s signature, which was created during the collection process.

6 Agent Migration

This section focuses on a distributed evidence collection scenario by illustrating the possibilities of agent migration. Wasting system resources on not currently needed services (e.g. agents) can be avoided using the JADE API and/or the JADE GUI for manual agent orchestration [2]. Figure 4 depicts the scenario of a multi/inter CSP agent distribution on demand: According to a given policy, the `A-PPL Engine (Accountable Privacy Policy Language)` enforces accountability policies, like planned in the `A4Cloud Project` [19].

Also, the reaction on an occurred event is described (e.g., the notification of a subject about the analyzed evidence scoring a specific result). The `A-PPL` is currently work in progress. It extends `PPL` which extends the `eXtensible Access Control Markup Language (XACML)` [20] which is why this part is currently emulated inside this architecture using a `XACML Parser` deciding whether a function will be executed or not (e.g. the migration function).

Because of the `A-PPL Engine`'s SaaS aspects [21] the service probably will run inside a different ISP's virtual machine, which also possibly will be stored

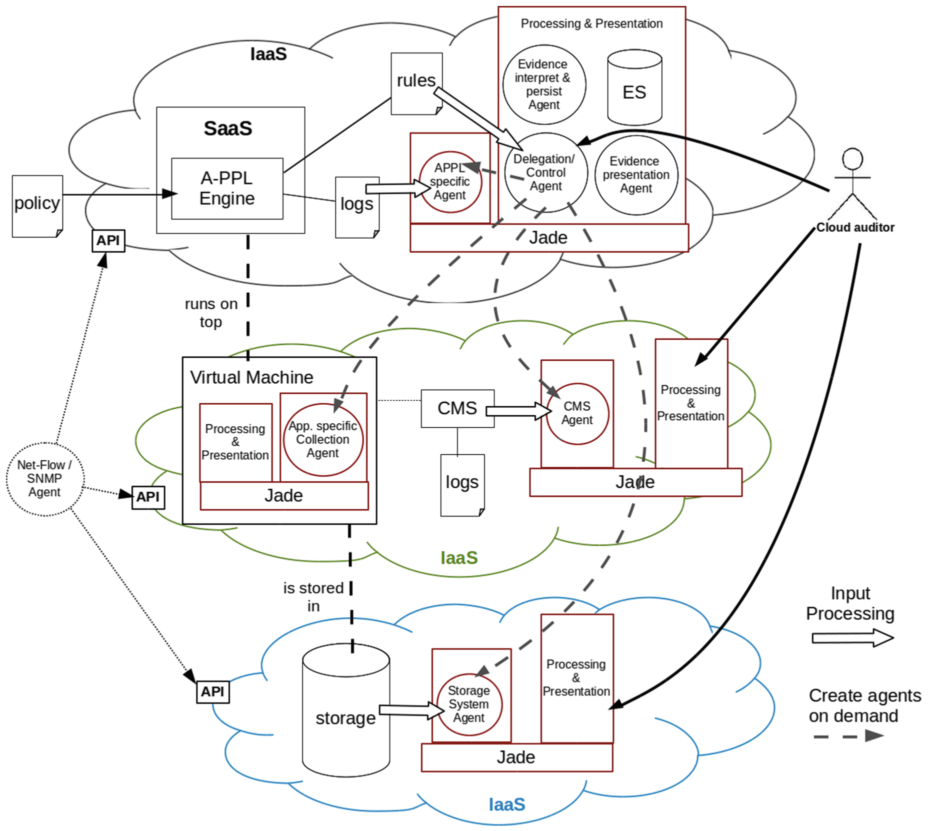

Fig. 4. Agent distribution on demand

inside a different ISP's Infrastructure. To provide a high availability the data is probably replicated to another data center, which possibly is located in another country and because of that following another juristic system [22]. Therefore, the actual avail of transparent data location in the cloud is to interpret as additional risk for a costumer.

Depending on the imported rules the `Delegation/Control Agent` distributes/ migrates the requested `Evidence Collection Agents` to their JADE destination platform. The different scarcely spawned `Evidence Collection Agents` initially performing a service registration after noticing their corresponding `Processing Layer` to communicate with. To avoid unnecessary overloading data traffic the evidence is always persisted inside their corresponding `Evidence Store` (weak migration).

The possibilities of network analytics are given anyway if the `Net-Flow Agent` is positioned ISP-local. To implement the `Net-Flow Agent` in a CSP-comprehensive way, every CSP must offer either a standardized API to check out the necessary connection stats or executing a (continuous) `Net-Flow Agent` by their own to communicate with. Potentially this API will implement the `Cloud Trust Protocol (CTP)` [23]. To collect every available net-flow evidence the different `CSP`s must report their supporting services to the requesting system (e.g. this architecture) or respectively provide a way to check them out. While there is no standardized `API` the opportunity of using a `Net-Flow Agent` on every service providing `CSP` is given (see Sect. 7).

Also the possibility of temporary migration is given, where an agent will be transferred to another platform and migrated back to the ordering agents platform (strong migration). In this case, not only the agents executable is transmitted between the platforms but its currently objects containing new insights, too. Of course, distributed agents remove themselves from their corresponding platforms in case of a non-continuous evidence collection run. To collect evidence, the corresponding agent platform must be started with the required rights to access the resource (in most cases this is `root`).

7 Complex Service Provision Scenarios

A service provision chain scenario demands a correlation of evidence collected by all involved services and their platform's `Evidence Collection Agents`. The complexity of this service provision refers to the inclusion of multiple `Service Providers` and different companies, respectively.

Figure 5 depicts a scenario where `CSP A` is offering a (potentially public) `Service A` but using `Service B` provided by `CSP B` (transparent to costumers). In this example, a network communication with a country outside the EU takes place at the service provided by `CSP B`.

The agreed upon policy predicates that every processed data must been held inside the European Union but because of the supply chain CSP interoperability is needed, more precisely a trusted service is needed.

Both `CSP A` and `CSP B` are hosting services inside their own datacenters located inside the EU. All necessary evidence connections are provided using this evidence collection architecture trusting a central `Locality Compliance Agent` which is aware of all service relationships used by a potential costumer.

If `CSP A` respects the policy by hosting a service inside the EU but is in turn using a service provided by `CSP B`, `CSP A` alone is not able to guarantee the compliance to this policy.

Requesting evidence reports from the affected (distributed) platforms `Delegation/Control Agent` is how compounded evidence is come about. Depending on the data transmitted from `Service A` to `Service B`, potentially valuable information could be transmitted to a *'forbidden'* location, which is why the `Evidence Collection Agent` placed at `CSP B` diagnoses for network communication policy violations and creates an evidence record inside its platform's `Evidence DB`. Once the `Locality Compliance Agent` receives all necessary evidence reports from the participating `CSP`'s `Presentation Layers`, the occurrence of interdependent policy violations are checked depending on the given EU data policy. Every new insight about interdependent policy violations will be

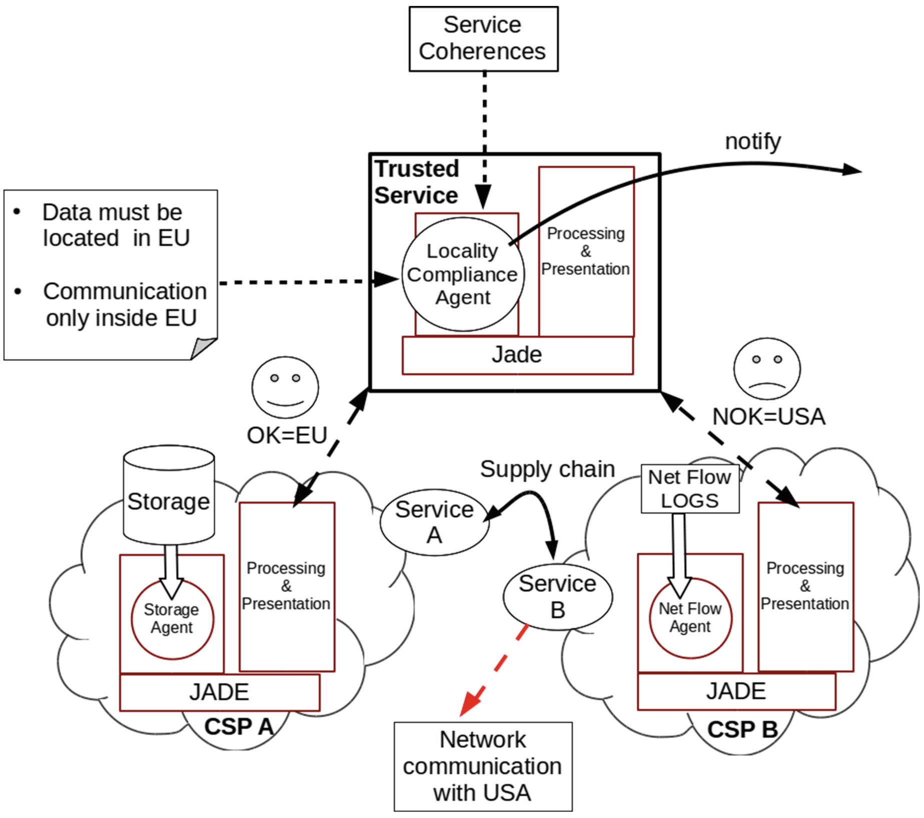
Service Coherences
notify
Trusted Service
Locality Compliance Agent
Processing & Presentation
Jade
• Data must be located in EU
• Communication only inside EU
OK=EU
NOK=USA
Supply chain
Storage
Storage Agent
Processing & Presentation
JADE
CSP A
Service A
Service B
Net Flow LOGS
Net Flow Agent
Processing & Presentation
JADE
CSP B
Network communication with USA

Fig. 5. Service coherences

forwarded to the `A-PPL Engine` which possibly will consider further steps (i.e., notifying stakeholders). Passing the new insights along the chain of accountability is relevant for a full conclusion but must go easy on network resources at the same time if this evidence architecture is applied in large datacenters. Therefore, for example a simple *'OK=EU'* or *'NOK=USA'* message on demand is sufficient for a pooled decision filling the dashboard of a customer's private web interface using the boolean product of all replies, group of replies respectively.

8 Conclusion and Future Work

This paper underlines the importance of structured evidence collection on demand running on nearly every device supporting transparency and therefore trust. The presented architecture enables distributed collecting and persisting of evidence following imported rules. Because of the possible distributed `JADE` platforms this construct will work in large `CSP` data centers without overloading the average performance.

To provide a quick evidence processing there must be a mechanism that excludes already known policy violations at `Evidence Collection Agent` side. The mentioned `Sealed Evidence Bag` [4] is a potentially evidence persistence extension. Also, the collected evidence must be presented in a convincing and distinct way at the user interface. Still existing challenges among other things are the performance evaluating of the defined architecture and scaling tests enabling the deployment of this architecture for highly dynamic services.

Acknowledgment. This research is closely related to the `A4Cloud Project`.

References

1. Reich, P.D.C., Rübsamen, M.S.T.: Supporting cloud accountability by collecting evidence using audit agents. In: 2013 IEEE International Conference on Cloud Computing Technology and Science (2013)
2. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology. Wiley, Chichester (2007)
3. Turner, P.: Unification of Digital Evidence from Disparate Sources (Digital Evidence Bags). QinetiQ
4. Schatz, B., Clark, A.: An open architecture for digital evidence integration. In: AusCERT Asia Pacific Information Technology Security Conference
5. Validating Cloud Infrastructure Changes by Cloud Audits
6. Nagios. www.nagios.com
7. New Relic. http://www.newrelic.com/
8. Cloud Control. www.cloudcontrol.com
9. App Fog. www.appfog.com
10. Italia, T.: Java Agent DEvelopement framework. http://jade.tilab.com
11. Poslad, S.: Specifying protocols for multi-agent systems interaction. ACM Trans. Auton. Adap. Syst. (TAAS) **2**(4), 1–24 (2007)
12. Foundation for Intelligent Physical Agents. http://www.fipa.org/
13. Reddy, P.I.P., Damodaram, D.A.: Implementation of Agent Based Dynamic Distributed Service
14. Nikraz, M., Caireb, G., Bahri, P.A.: A Methodology for the Analysis and Design of Multi-agent Systems using JADE. Telecom Italia Lab
15. Jansen, W., Grance, T.: Guidelines on security and privacy in public cloud computing. National Institute of Standards and Technology, U.S. Department of Commerce (2011)
16. H2 Database Engine. http://www.h2database.com
17. Open Nebula. http://opennebula.org/
18. Tchamdjou, M.Y.D.E.: Agenten zur Erkennung von sensiblen Daten und deren Schutz. HFU, Technical report
19. Accountability for the Cloud. http://www.a4cloud.eu/
20. XACML - Extensible Access Control Markup Language. www.oasis-open.org/
21. Benghabrit, W., Grall, H., Royer, J.-C., Sellami, M., Azraoui, M., Elkhiyaoui, K., Önen, M., Santana De Oliveira, A., Bernsmed, K.: A cloud accountability policy representation framework. In: CLOSER - 4th International Conference on Cloud Computing and Services Science, Barcelone, Espagne (2014). http://hal.inria.fr/hal-00941872

22. Bradshaw, S., Cunningham, A., Luciano, L.D.C., Hon, W.K., Hörnle, J., Reed, C., Walden, I. In: Millard, C. (ed.) Cloud Computing Law. Oxford University Press, Oxford (2013)
23. Cloud Trust Protocol. https://cloudsecurityalliance.org/research/ctp/