Application of Authorization in Smart Grid based on the PasS Microservice architecture

Ding Yi¹, Tong Donghui¹, Wang Xuegang¹, Zhang Shuliang¹*

¹Liaoning Electric Power Energy Development Group Co., Ltd. Shenyang, China

Corresponding author’s e-mail: sg_eiconf@126.com

Abstract. This paper introduces a smart grid environment authorization service architecture based on XACML and SAML standards. Software prototypes are deployed with the platform as a service (PaaS) framework Appscale. Appscale is an open source AppEngine framework released by Google that can execute applications in the cloud to solve the problems caused by the distributed computing model. This paper mainly evaluates its scalability. Through the different configuration of all distributed databases, the test of the system running load is completed. The test results show that the number of nodes deployed by Appscale varies from very unstable behavior to waiting scalability.

1. Introduction

Smart grid is a combination of digital processing and communication applications. Smart grids support collaboration between manufacturers, storage facilities, and users to improve the efficiency of grid operations.

Figure 1 shows a demonstration of the smart grid service application. As you can see, the ICT equipment for the application service is located at the connection point of the grid, and smart meters are installed at the end user’s site. These meters can remotely access the user's power consumption data, which can be used to assess how much power is actually needed. Home automation services make it possible to adjust household electricity consumption to different energy prices. Additional services for transformer monitoring, distribution automation, public lighting and workforce automation are also further enriching the smart grid function.

![Figure 1. Services applied in a smart grid environment](image-url)

Different application services generate a large amount of data, which can be applied to different industries through analysis. In order to complete the distribution of massive amounts of information, you need an interface that is available to all participants. Web service standard is a widely used interface...
with the advantages of platform independence, interoperability and operability. At the same time, Web services have a good ecosystem, supported by many enterprise platforms to create and use. Because Web services are open, authentication and authorization are their service stack issues. The data generated by service application in smart grid is highly sensitive. If the intruder gains access, he can analyze the application data and obtain other information. Therefore, it is necessary to let the user who provides the data set access rights.

In order to ensure data privacy for all services used in the smart grid, this paper introduces and evaluates an authorization system for service applications that can file configure XACML-SAML.

2. Authorization as a Service architecture

It is proposed a standard architecture based on XACML[1] and SAML[2]. These standards provide a policy language and an access control decision request/response language. In addition, a workflow model is defined to enforce application policies on multiple components. The standard requires that requests to access resources be handled by a policy enforcement point (PEP) that acts as an interface to the outside world. This component is an authorization decision making request processed by a policy decision point (PDP). The PDP evaluates the request based on the available policies and returns the authorization decision to the PEP. This component then handles the initial request. In our architecture, additional identity components are responsible for user authentication.

Figure 1 is an access control system architecture diagram. The executing PEP component is located near the service that needs to be protected. The identity component (identity provider (IDP), Identity Store (Identity Store), and evaluation component (PDP) reside in a centralized cloud infrastructure. They are responsible for handling the access control tasks associated with all smart grid services. In a given domain, the following actors need to be mapped to the language elements defined by the XACML standard for policy formulation:

- End users who have devices connected to the smart grid, such as smart meters;
- Web services provided by the devices;
- Applications that wish to access these Web services;
- Operations of Web services that can be invoked.

Model according to system rules, for example, "some end user A provides Web service B and allows an application C to invoke operation D. "Therefore, the end user is defined as a Web service data resource. A particular end user matches a unique id id, modeled as a resource id. A Web service has the property of service Name. According to this structure rule, a policy set can be explicitly identified by a combination of the resource ID and service name of the policy it contains. The application is the primary modeling object, represented by subject-id. The system provides policy support based on a single topic identified by subject-id. The permitted operation (SOAP operation) of a Web service can then be modeled in the form of permitted operation properties.

3. Proof of Concept

Scalability and performance are important issues in future deployment of smart grid scenarios. PDP and PAP components have been implemented and migrated to Appscale [3]. Appscale USES platform-as-a-service (PaaS) open source technology to implement the Google App Engine (GAE) application programming interface (API). For better data storage, you can use Appscale's distributed database feature. Because the API provides a public interface that is independent of the underlying database instance, you can test different distributed database technologies without changing the code. These databases have been "out-of-the-box" without further tuning.

3.1. PDP component

In order to communicate with the outside world, the PDP component publishes a Web service that produces a specific interface specified in the XACML-SAML profile. Since GAEAPI only supports the use of servlets, SOAP requests must be processed separately. The transformed request is then forwarded to the component that evaluates it according to the rules defined by the XACML standard. To prove that
this approach works, the parsing process of the received data messages must be tuned so that GAE can use a whitelist of Java classes during use. The introduction of the whitelist mechanism is to protect the GAE environment from unsafe factors.

3.2. PAP component

The PAP component was developed with the support of Appscale's distributed database. Exchange of application data through a Web service and a Web form. The Datastore Service provided by GAE can complete the storage and service retrieval strategy of the application interactive data. Key properties and policies in plain text are added as properties of the policy store. To retrieve the policy, you must provide the key properties. They are then used to create a query that results in an appropriate retrieval strategy. After the first round of testing, a second implementation for policy retrieval was created in conjunction with the memcache service. It is used to cache policy data retrieved from the database to save processing time and prevent future requests for the same policy.

3.3. Policy Store

In the policy stored procedure, the primary keys for recording are the key attribute resource Id and the service name. GAE API USES entities stored in the database, grouped by the type specified during creation. They can have one or more named attributes, each of which can have one or more values. Homogeneous entities do not need to have the same attributes. Based on these facts, the policies described above are stored in entities that have such policies. It has an attribute resource Id, service name, and policy content. The first two contain corresponding key properties that can be used to query the entity.

Apart from just porting Picketlink to Appscale the policy lookup mechanism was changed significantly. In its original version, Picketlink bootstraps its PDP with all available policies. These are loaded and kept in memory through the whole runtime of the program. Since high amounts of policies with the same structure are anticipated in our use case this method was not very practical. Therefore, the Policy Lookup mechanism was tweaked to conform with the mechanism proposed in the designed architecture. When evaluating a received request, Picketlink asks a PolicyFinder to return the right policy. Normally, Picketlink would search its cached policies for the one with the matching targets. In our implementation, the database based PAP is queried for the right policy by supplying the key attributes that have been extracted from the request. The resulting policy is then returned to the PolicyFinder.

4. Evaluation

Figure 2 shows the underlying cloud infrastructure architecture for the XEN, the virtualization framework that applies to multiple hosts, allows the creation of virtual machines that can be flexibly extended according to their specific performance requirements. Appscale is deployed on these virtual machines. The distribution of incoming requests between the component work nodes is completed by the load balancing node. The database node is responsible for completing access to the database. The PDP and PAP components are deployed on this architecture. With the API components provided by Appscale, you don't need to deal with a particular cloud task.

During test execution, there were only dedicated computers in the network for the JMeter test. During the test routine, the computer invokes the Appscale application and completes the specified test plan. Appscale performs tasks on XEN servers of multiple virtual machines at the same time. Each database provided by Appscale (with the exception of the MySQL Cluster) has been tested in several ways, and the deployment scope has been expanded from one node to up to four nodes. The MySQL cluster configuration requires that the total number of nodes be replicated continuously, meaning that there are six nodes and two or three replicates are possible.
The purpose of closed-loop load testing is to issue a decision request to the PDP interface of the access control system to check the effectiveness of the system. Request instructions are created in a CSV file containing 10,000 randomly generated unique ids. During the test, requests are sent alternately, and the test results are judged by the return permission or reject instructions. The load generated by the test is controlled by the number of threads making the request. The number of threads increases every five minutes until the maximum count is 36. The maximum load was maintained for 5 minutes until the end of the test run.

Table 1 shows the closed-loop test results. Compared with the results of single-node Appscale deployment test, the performance of multi-node deployment test is obviously reduced. The reason for this is that the single-node setup does not require the balancing of the load, just putting all the application components on one node and not doing complex monitoring logic. Hypertable and Cassandra show the "service unavailable" response. During the single-node Appscale deployment test, once the request is queued and reaches 60 seconds, a "service unavailable" response is triggered. In all the databases tested by Appscale, the MySQL cluster performed best. The load created by the test run does not appear to be "service unavailable." MySQL cluster deployment improves its performance from two nodes to three nodes, showing higher performance than single node deployment. In addition, MySQL clusters are also less expensive to perform replication tasks than other databases. Appscale seems to provide much better scalability in MySQL clusters than other databases.

Table 1. The closed-loop test results

| Node(s) | Hypertable | Cassandra | MySQL Cluster |
|---------|------------|-----------|---------------|
| 1       | 22 tpm     | 44 tpm    | 185 tpm       |
| 2       | 5 tpm      | 19 tpm    | 178 tpm       |
| 3       | 11 tpm     | 22 tpm    | 325 tpm       |
| 4       | 16 tpm     | 27 tpm    | ---           |

The other databases not listed in table 1 (HBase, MemcacheDB, Voldemort, MongoDB) did not return any available results. The Appscale developers stated on the mailing list that these databases did not pass much test time.

In order to get more information about the data received in the executed tests, the actual processing time of the AppServer component was analyzed. By recording time Settings on the program code, you can measure the time it takes to execute Java code to process requests. The differences between the databases used and the time span of query database are further studied. Table 2 shows the aggregated data for the MySQL Cluster and Cassandra databases. As you can see, the processing time of the MySQL cluster is almost the same under various configurations. If multiple nodes are used in an Appscale deployment, Cassandra's processing time increases significantly. More importantly, the database access time for the MySQL cluster is less than 1 second, while the database access time for Cassandra is 7 to 14 seconds. The MySQL cluster performed better than the Cassandra results.
Table 2. AppServer processing times

| Node(s) | MySQL Cluster | Cassandra |
|---------|---------------|-----------|
|         | DB Acc. | Full Req. | DB Acc. | Full Req. |
| 1       | 0.2 sec | 0.96 sec  | 7.26 sec | 8.13 sec  |
| 2       | 0.32 sec | 0.99 sec  | 8.67 sec | 9.56 sec  |
| 3       | 0.23 sec | 0.99 sec  | 14.66 sec | 15.85 sec |
| 4       | --      | --        | 14.27 sec | 15.44 sec |

5. Conclusion

Appscale provides the possibility of specifying the location of each Appscale component through a configuration file. To keep the test count at a viable number, only the standard location of the Appscale component is used. In further testing, you can assess a number of positional changes that may have a significant impact on the resulting performance. Efficient XML exchange (EXI) is a binary XML format for encoding XML documents. You can further improve performance by reducing the size of the messages that must be sent between the requester and the PDP component. Because the architecture handles incoming and outgoing SOAP messages, it is possible to integrate the technology directly without changing the entire architecture.

References

[1] eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS Standard, Feb. 2005.
[2] Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, OASIS Standard, Mar. 2005.
[3] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In Cloud Computing, volume 34 of LNCS, chapter 4, pages 57–70. Berlin, Heidelberg, 2010.
[4] V. C. Gungor, D. Sahin, T. Kocak, S. Ergt, C. Buccella, C. Cecati, and G. P. Hancke. Smart grid technologies: Communication technologies and standards. IEEE Trans. Indus- trial Informatics, 7(4):529–539, 2011.