

X.509 Authentication/Authorization in FermiCloud

Hyunwoo Kim, Steven C. Timm

Scientific Computing Division
Fermi National Accelerator Laboratory
Batavia, U.S.A
hyunwoo@fnal.gov, timm@fnal.gov

Abstract—We present a summary of how X.509 authentication and authorization are used with OpenNebula in FermiCloud. We also describe a history of why the X.509 authentication was needed in FermiCloud, and review X.509 authorization options, both internal and external to OpenNebula. We show how these options can be and have been used to successfully run scientific workflows on federated clouds, which include OpenNebula on FermiCloud and Amazon Web Services as well as other community clouds. We also outline federation options being used by other commercial and open-source clouds and cloud research projects.

Keywords—Cloud; X.509; Authentication; Authorization; FermiCloud

I. INTRODUCTION

A. X.509 Certificates For Identity Authentication

FermiCloud relies on X.509 certificates [1] to achieve identity authentication. X.509 provides us with a way to verify the user’s identity is in fact who he or she claims to be. The identity of a user can be verified by a chain of signing authorities.

Three basic concepts, identification, authentication and authorization, must be considered with equal importance in order to make sure the right users are doing the right things in our system. Identification is how users assert who they are to our system. It can be your user name if the system is relying on username and password authentication. Authentication is how users prove their identity assertion. In other words, it is a presentation of secret that only the owner must know and the system can then verify the identity with. If it is password, the server should keep the secret and use it to verify the user when the user types in the password when the user wants to sign in. Furthermore all communications can be encrypted by using SSL. Since it can be presumed that only a user knows his or her secret, the user’s claim can be validated by authentication.

Identification and authentication in X.509 scheme are closely related. Identity in X.509 scheme can be the subject name on a X.509 digital certificate. In analogy with username and password scheme, the secret could be the private key associated with a digital certificate. The difference between username and password scheme and X.509 scheme is how

users present the secret. With the simple username and password scheme, as described above, the server must keep the secret and the user has to type in the password in a browser. In the X.509 scheme, users present their Distinguished Name when their account is created. Each time they authenticate, they present the full certificate and private key credential, using the passphrase to decrypt the private key. The login process will sign a simple text (username in case of OpenNebula) with the private key and transmit this private-key-signed text to the server. The server verifies this text using X.509 certificate that came along with the document and then extract the DN from the X.509 certificate. Then the server compares this DN against the list of DNs stored in the server database. The process described above is one of basic cryptographic assurance that is provided by public key cryptography. In principle, with RSA[2], we can use private key to encrypt the entire plaintext. This provides both identity authentication and message authentication because only the person that holds the private key can encrypt a document and because that person alone could guarantee the authenticity of a document by encrypting the entire document. What are transmitted are the plaintext and the encryption result. While PKC uses public and private keys to achieve cryptographic assurances, we still need to prove the ownership of the public key, which should be distributed in a manageable way. In other words, public key must be distributed as digital certificates. Public key infrastructure (PKI) is one way to achieve this and X.509 is an ITU-T standard for PKI [16].

B. Authorization with X.509 Certificates

Authentication on its own is not sufficient to allow users to use resources. We also need to know what actions, if any, a user is authorized to undertake. The existing OpenNebula[13] authorization scheme is based on Access Control Lists for resources. All resources in OpenNebula, including virtual networks, machine images, and templates, have user, group, and world permissions similar to Unix permissions. There are also per-user and per-group quotas of how many machines can be launched.

FermiCloud is developing a new X.509 based authorization module for OpenNebula that also can determine the correct user and group given a grid identity. It uses information from Virtual Organization Membership Service (VOMS) [3] and Grid User Management System (GUMS) [4]. With a user’s subject line in X.509 certificate, we first contact a local

VOMS in FermiCloud to acquire a list of Fully Qualified Attributes Name (FQAN) assigned to that user. We present this list to the user prompting for the user's selection. Then we construct an XACML (eXtensible Access Control Markup Language) [5] request using an XACML Java client library called privilege package, developed at Fermilab, and send this request to a local GUMS server. We expect two answers from GUMS: a pair of FermiCloud-specific user ID and group ID mapped by GUMS and secondly whether the user is authorized to undertake the Role and Capability in the FQAN. We use this information to finally authorize the user and also record which VO a virtual machine is running under the name of. Technical details are found in a later section.

C. History of why X.509 Authentication was needed

X.509 based authentication and authorization became widespread in the scientific community with the widespread adoption of grid computing. The Grid Security Infrastructure (GSI) [6] includes a set of certificate authorities that are recognized by the International Grid Trust Federation and accepted worldwide by grid computing sites. All Fermilab-hosted experiments participate in grid computing via the FermiGrid [7] campus grid and the Open Science Grid [8] of which FermiGrid is a major part. When the FermiCloud [17] project was initiated in 2009 we identified a requirement to have stronger security than the default username/password or access key / secret key mechanism could provide. We had several years of successful operation of X.509 authentication and authorization on the grid and an interoperability protocol for authorization based on XACML authorization[18] which we hoped to reuse. By using X.509 authentication and authorization we could know exactly who is running virtual machines on our cloud. An X.509 authorization scheme also allows us to transparently let a user with a single X.509 Distinguished Name be part of more than one group or organization, and transparently change between them. Because Fermilab operates its own Short Lived Credential Service (SLCS) certificate authority we can further auto-generate short-lived certificates on behalf of our own users and revoke them at any point. In practice much of the X.509 certificate manipulation is transparent to the user and invoked in the login script as they log into the interface node.

II. OPENNEBULA IMPLEMENTATION OF X.509

A. Token-based Authentication in OpenNebula before using X.509

Besides X.509 authentication, OpenNebula also provides another token-based authentication method that uses ssh keys. When a new user is signed up, the new user has to use ssh-keygen command to generate a pair of ssh keys and register the public ssh key in OpenNebula system. For sign-in, the regular login command with the option of using ssh keys will create a Single Sign On (SSO) token and this token will be used for subsequent uses of user commands.

B. X.509 Authentication in Command Line Interface

FermiCloud developed X.509 module for OpenNebula. The basic idea is token-based SSO authentication using user's X.509 certificate and associated private key. A user initially executes a login command with X.509 certificate and private key. Internally this command uses the user's private key to sign a text document, base64-encodes it and produce eventually a token as a secure file in the user's private area. Subsequent commands issued by the user will present this token to the OpenNebula server. The server first retrieves the user's X.509 certificate and uses it to verify (authenticate) the identity of the user. This implementation was incorporated into the main OpenNebula 3.0 code base in 2012 and has continued to be available since that time in all subsequent versions.

FermiCloud implementation of X.509 authentication follows a common approach to achieve X.509-based authentication. We note that this is also the case for OpenStack PKI-signed token-based authentication that we will review in a later section.

1. An X.509 authentication scheme must provide a tool a user can use to sign in. This command will require both X.509 certificate and associated private key from a user. The command then will generate a token and sign it with the private key.
2. The server side, first of all, should be able to acquire the user's X.509 certificate. In the above description, the client tool requires the user's X.509 certificate too besides the private key. In case of OpenNebula, the client tool appends the X.509 certificate to the signed token. In case of OpenStack, each service endpoint (such as Nova) downloads a X.509 certificate from a pre-defined location.
3. The next question is, how to transmit this signed token to the server side. OpenNebula CLI generates a token in a form of a file. The user must set an environment variable to the location of this file so that the OpenNebula command line tools can transmit the appropriate authorization data to the "oned" daemon. As OpenStack only supports RESTful services (via direct use of URL in a tool such as cURL, OpenStack SDK or OpenStack CLI), OpenStack adopts a different method for a transmission of the signed token and use X-Auth-Token HTTP header for this purpose.
4. The server side can conduct a simple verify operation against a private-key-signed token with a X.509 certificate of OpenNebula user or the Keystone signing certificate in the case of OpenStack.

After verification, OpenNebula extracts the DN of the user from X.509 certificate and uses it to identify the user against the list of Distinguished Names stored in the user pool table of the database. OpenStack simply uses the username for identification. As mentioned above, this scheme is used by X.509 authentication of OpenNebula CLI and OpenStack

Keystone PKI-based token authentication. Note that the scheme described above cannot be done with the normal use of web browser because the browser doesn't allow the same flexibility as CLI utilities of using user certificate and private key for cryptographical purposes.

C. X.509 Authentication in Sunstone OpenNebula Web Interface

OpenNebula Sunstone is basically a Sinatra-based web application with Thin [9] in front of it as a Ruby web browser. Sunstone is usually placed behind Apache HTTPD with SSL module. There are two types of clients that will access Sunstone. Users can upload their certificate and private key pair to web browsers to access REST services provided by Sunstone. Web browsers allow only PKCS12 [10] format and require the encryption password when the certificate and key are imported into the browser. The certificate is then protected by the password of the certificate store of the browser, which must be given when the certificate is used at a given site for the first time. The Sunstone service and other such services can also be accessed with CLI tools such as the cURL command. The cURL command for example also requires the users to provide both private key and certificate at the same time and prompts for the password (if any) associated with the private key. The cURL command also goes through SSL handshake with the SSL module attached to Apache server that is in front of Sunstone. In both cases, the SSL module for Apache on the server side is configured to use the option to verify client for a client-authentication. After the SSL handshake finishes successfully, the SSL module sets an environment variable called `HTTP_SSL_CLIENT_CERT` equal to the base-64 encoded PEM string of the full client certificate that was transmitted from the browser or the CLI tool. Sunstone code uses the X.509 Certificate utility in Ruby OpenSSL library in order to extract the user's Distinguished Name from the PEM string and uses the extracted DN to compare against a list of DN's in the user pool in order to acquire the username that matches the DN. This look-up process is common to both username-password and X.509 certificate schemes, but using X.509 certificate, we can identify a user who is verified by Certificate Authority (CA). We note that this authentication scheme in OpenNebula Sunstone is functionally similar to the external authentication option that OpenStack supports besides the regular methods using username-password or token.

D. X.509 authentication in EC2 emulation

OpenNebula EC2 emulation is also a Sinatra application with Thin web server behind Apache HTTPD with SSL module. Current implementation of OpenNebula EC2 emulation client code is using AWS Ruby SDK to generate REST/Query requests to OpenNebula EC2 emulation server using AWS signature algorithm to sign the request with Access Key ID and Secret Access Key. FermiCloud modifies OpenNebula EC2 emulation codes in order to enable the use of X.509 certificates. We replace AWS EC2 library with Ruby cURL library to generate REST requests. This way we can

exploit the options for X.509 certificate and private key available in the Ruby cURL library and achieve the same client authentication as what happens between the cURL command and the OpenNebula Sunstone. In current implementation of OpenNebula, the EC2 emulation server shares the same X.509 authentication code with the Sunstone. Apache with SSL module processes and forwards secure information to EC2 emulation Sinatra server. EC2 emulation server uses this information to reconstruct the user's X.509 certificate with X.509 Certificate utility in Ruby OpenSSL library and extracts the user's DN to compare against the list of DN's in user pool. It was necessary to modify the code slightly to allow the server to accept X.509 proxy certificates as well as full certificates.

E. X.509 Authorization developed by FermiCloud

The existing OpenNebula authorization is based on Access Control List. After a user is authenticated, relevant access control information or permissions related to the user are examined to determine the authorization results. We started by modifying CLI to use Local Credential Mapping Service (LCMAPS) [11] developed by NIKHEF. When a user signs in with OpenNebula CLI, both proxy certificate PEM string and personal certificate PEM string are available in OpenNebula server side where we call LCMAPS C function via Ruby C binding in order to contact FermiCloud GUMS server. Then, we extended this solution to Sunstone. The fact that we need both proxy and personal certificates to call LCMAPS function means that we need to plant both proxy certificate and personal certificate into the browser, which is technically possible. The problem was with the transmission of certificates from web browsers to the server. The proxy certificate is available in the server side as a PEM string, but the personal certificate that was used to generate the proxy certificate is not transmitted. Technically this is believed to originate from the fact that web browsers do not recognize the personal certificate as a proper CA that signed the proxy certificate. Using the cURL command, the server sets these variables properly. For this reason, we do not consider using LCMAPS to contact GUMS because it does not work with the web browsers. We decided to use an XACML client library to build an XACML request. This client library is called privilege package and was developed in Java programming language by Fermilab. In order to invoke this Java privilege package from a Ruby code, we use Ruby Java Bridge (RJB). A request to GUMS requires user's DN, VO and FQAN. Our solution to acquire user's VO and FQAN is contacting VOMS-Admin server in FermiCloud. We then present this list to the user asking for the user's selection. Then with the selected VO and FQAN, Sunstone constructs a request to GUMS using privilege package. The query to VOMS-Admin for all the possible groups and roles has the benefit that we can use a grid proxy or certificate rather than one with extended VOMS attributes and these are easier to manage in the browser. This successful implementation in Sunstone could also be applied to command line and EC2 emulation interfaces of OpenNebula.

We also tried to use GridSite package as a module for Apache HTTPD. GridSite can be used to retrieve the VO and FQAN from a VOMS-signed certificate that is transmitted from a web browser. In our testing, GridSite package worked properly only when cURL command was used and it failed when web browser was used.

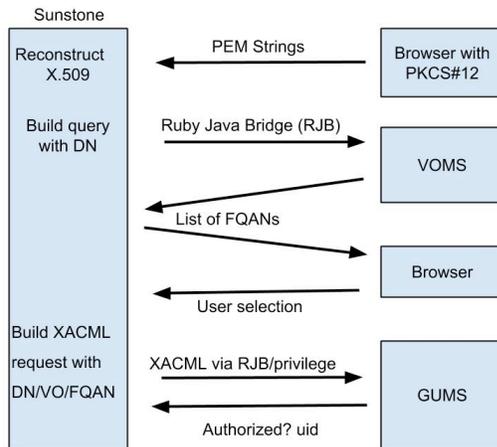


Figure 1. FermiCloud new Authorization Scheme

F. Scientific Workflows on Federated Clouds

Fermilab uses the GlideinWMS [19] workflow management system and the Jobsub client/server submission system to federate heterogeneous resources including grids and clouds. Users use their automatically generated SLCS-based X.509 certificates to authenticate to the server and submit jobs. The GlideinWMS system uses its own certificates and AWS access keys to obtain grid job slots and virtual machines on behalf of the federation of users, and then these resources are matched to user jobs based on the requirements of these jobs. We use this infrastructure to support Fermilab physics experiments. We have successfully run the Cosmic Ray simulation of the NOvA neutrino detectors on Amazon AWS services, FermiCloud, and a collection of sites on the Open Science Grid. We continue to increase the amount of virtual machines we can run simultaneously.

III. X.509 IN AMAZON WEB SERVICES EC2

AWS basically supports two APIs: REST and SOAP. If we want to use SOAP API, there are two possible ways. We can use AWS EC2 SDK to generate SOAP requests or we can use AWS EC2 CLI to generate SOAP requests. In both cases, we can use X.509 certificate and private key for authentication. AWS will discontinue the support of SOAP APIs at the end of 2014. If we want to use REST API, there are three ways. We can access REST API directly by constructing a query with a command line tool. Or we can use AWS EC2 SDK or CLI to generate Query requests. The Amazon EC2 REST API provides HTTP or HTTPS requests that use HTTP GET or POST methods and a Query parameter named Action. These AWS REST requests are signed using Access Keys that

consist of Access Key ID and Secret Access Key. Note that the AWS Command Line Interface or the AWS SDKs automatically sign requests for us. But if we construct a Query request directly, we must sign the requests manually, using the procedure described in AWS signing algorithm.

Also we note that there are several AWS credentials types for different purposes.

1. Email address and password: when we sign up for AWS, we provide an email address and password that is associated with our AWS account. We use these credentials to sign in to secure AWS web pages.
2. Access Keys: we use access keys to sign requests to AWS whether we're accessing the REST API via the AWS SDK, CLI or direct access.
3. X.509 Certificates: we are recommended to use X.509 certificates only to sign SOAP-based requests. In all other cases, we are recommended to use access keys.
4. Key Pairs: for Amazon EC2, we use key pairs to access Amazon EC2 instances, such as when we use SSH to log in to a Linux instance.

IV. EGI FEDERATED CLOUD

European Grid Infrastructure (EGI) [12] recently launched a project called Federated Cloud (FC). General structure of EGI FC is rOCCI server in front of cloud resources using OpenNebula or OpenStack. The rOCCI server consists of a Rails web application and Apache HTTPD with SSL module and Passenger [12] module. We can issue an occi client command with options for X.509 authentication and the use of VOMS. We are interested in how the rOCCI OpenNebula backend conducts X.509 authentication. The backend invokes OpenNebula UserPool method that is available in a local OpenNebula distribution that resides in rOCCI server deployment. This UserPool contacts the main OpenNebula instance via the regular xml-rpc channel to receive a list of users that are registered. Then local methods in the rOCCI OpenNebula backend such as do_auth will see if a valid username is returned from a query using the regular X.509 DN when auth x509 option is used with occi command or using the extended DN with FQAN when auth x509 and VOMS options are used together in occi command. When the OpenNebula instance that is running behind rOCCI server is needed to support rOCCI's VOMS authentication type, each user should be created with a DN extended with FQAN. This can be done manually or by using Perun script. In details, the X.509 based authentication that is conducted by rOCCI OpenNebula backend relies on the list of users returned from the actual OpenNebula instance running behind rOCCI server and is equivalent to what the ordinary OpenNebula does for X.509 authentication. We are also interested in understanding how Perun is used in association with rOCCI's VOMS authentication. As mentioned in the previous paragraph, a user can be created with a DN extended with FQAN by using Perun script. This script first contacts a Perun server to retrieve an up-to-date list of users and associated virtual

organizations and accordingly update OpenNebula's user pool with the list and this update process will create a user, if necessary, with an extended DN with FQAN.

We note that the authentication and authorization model in EGI Federated Cloud is using information from VOMS only for authentication purpose and we will still need our new development for X.509 authorization even if FermiCloud OpenNebula is placed behind rOCCI server.

V. OPENSTACK AND FEDERATION

First of all, we note that both AWS and OpenStack support only RESTful Web Services. There are two ways to access RESTful Web Services. In a direct way we can use cURL or other external RESTful clients. Here, we need to construct the request for ourselves and interpret the raw response of XML or JSON. In an indirect way we can use SDK or CLI. They both need endpoint URL. They will both access the URL just like the direct way, but the difference is that the indirect way via SDK or CLI will process the raw response of XML or JSON and returned a formatted response. Any statements with regard to OpenStack refer to OpenStack version Icehouse, the current version at the writing of this paper.

A. Regular Authentication in Keystone

OpenStack consists of several Services. Keystone is the one that handles the Identity Service. Another example is Nova that handles the Compute Service. Keystone supports four authentication plugins, which are specified in the [auth] section of the configuration file: password, token, external and federation. Suppose a user has obtained a credential, i.e. username and password. This user might use username and password to issue a Identity API request of a token to the Keystone. Or if this user already has a token that is still valid, the user could use the token to issue a new Identity API request to Keystone. In either case, after a token is acquired, the user can use this token to issue subsequent API requests such as Compute API requests to the NOVA service. This is Single Sign On. The user can also still use username and password, to issue subsequent API requests.. This use of username and password as authentication method in Identity service and other service such as Compute is similar to the use of Access Keys in Amazon Web Services. When a token is used, Keystone uses PKI to sign and verify the tokens. Further, Keystone uses SSL to encrypt the communications. Use of token is similar to X.509 authentication in OpenNebula.

B. External Authentication

Web servers like Apache HTTPD support many methods of authentication. When Keystone is executed in a web server like Apache HTTPD, Keystone can profit from this feature and let the authentication be done in the web server. When a web server is in charge of authentication, it is normally possible to set the REMOTE_USER environment variable so that it can be used in the underlying application (Keystone). Keystone can be configured to use that environment variable

if set. This user must exist in advance in the identity backend to get a token from the controller. This way, we can use X.509 authentication or Kerberos, for example, instead of using the username and password combination. To use this method, Keystone should be running on HTTPD and Apache should be configured to enable SSL. Note that while it is possible to use an external authentication method besides password and tokens, it is also possible to use an external method for identity provider besides the SQL database backend. Popular choice is LDAP directory service.

C. OpenStack and Federation

Keystone can be placed behind Apache HTTPD for two reasons: to use external authentication method, besides the ordinary two methods, which are password and token. Second reason is to use federation. New feature in the latest version of OpenStack is Identity Federation using SAML [14]. External users authenticate with Identity Provider (IdP). The IdP communicates the authentication result to Keystone using SAML assertions. Keystone maps the SAML assertions to Keystone user groups and assignments created in Keystone. In order to make this possible, Keystone should be configured accordingly. First, Keystone should be driven by Apache httpd and Shibboleth should be installed. Secondly, Shibboleth itself should be configured. Third, there is an extension called OS-FEDERATION. This should be enabled. Lastly OS-FEDERATION extension should be configured.

D. Authorization in OpenStack Keystone

Role is a personality that a user assumes when performing a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges. In OpenStack Identity, a token that is issued to a user includes the list of roles that user can assume. Services that are being called by that user determine how they interpret the set of roles a user has and to which operations or resources each role grants access. It is up to individual services such as the Compute service and Image service to assign meaning to these roles. As far as the Identity service is concerned, a role is an arbitrary name assigned by the user.

VI. NIMBUS AUTHENTICATION

The Nimbus project [15] implemented X.509 authentication and authorization using a full Grid Security Infrastructure system. This included a WSRF (Web Services Resource Framework) application container based on the Globus toolkit, which was capable of authentication and authorizations using certificate/key pairs or proxies. There was also a gridftp-based image transfer service used by the cloud client. The project also later added emulations of the SOAP and REST API's of Amazon EC2 as well as the Cumulus storage element, which emulates the Amazon S3 API.

VII. SUMMARY

We learned from this study that X.509 authentication in SOAP based API is decreasing in popularity and RESTful API is the current trend. We reviewed how FermiCloud developed X.509 authentication for OpenNebula command line interface. The idea is similar to how OpenStack is using PKI-based token for single-sign-on type authentication in REST requests. This approach is one way to support X.509 based authentication in RESTful services whereas using username and password in RESTful services is dominantly popular as in AWS. We have also shown a proof of principle of a unified procedure based on callouts to VOMS-Admin and GUMS for X.509-based authorization in OpenNebula. We also reviewed how OpenNebula Sunstone is using X.509 authentication via Apache HTTPD with SSL module. This is also similar to how OpenStack is using Apache HTTPD with SSL module as an option for external authentication. We also reviewed how EGI Federated Cloud is using rOCCI to federate cloud facilities using OpenNebula and OpenStack and how OpenStack is using SAML based external identity provider to federate users with external identities. Our plan is to keep looking for the best way to achieve authentication and authorization on top of restful cloud services as many new concepts and technologies are being developed and made available publicly.

ACKNOWLEDGMENT

We thank the developers of OpenNebula for their continued cooperation in adding authentication and authorization features that we have requested. We also acknowledge the significant contribution of former Fermilab employee Ted Hesselroth who was a member of the project through the fall of 2011 and was largely responsible for the X.509 authentication code that was contributed to OpenNebula. This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359 and by KISTI under a joint Cooperative Research and Development Agreement. CRADA-FRA 2014-0002/KISTI-C14014.

REFERENCES

- [1] R.Hously et al, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile" <https://www.ietf.org/rfc/rfc2459>
- [2] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signature and Public-key Cryptosystems", *Communications of the ACM* 21 120-126 1978.
- [3] R. Alfieri et al. 2004. VOMS, an authorization system for virtual organizations *Proceedings of European across Grids conference No1, Santiago De Compostela, Spain 2970* 33-40
- [4] M. Lorch, D. Kafura, I. Fisk, K. Keahey, G. Carcassi, T. Freeman, T. Peremutov, A. S. Rana. 2005. Authorization and account management in the Open Science Grid *The 6th IEEE/ACM International Workshop on Grid Computing, 2005*
- [5] <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cos01-en.html>
- [6] <http://toolkit.globus.org/toolkit/security/>
- [7] <http://fermigrid.fnal.gov>
- [8] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wurthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick 2007. The Open Science Grid *Journal of Physics: Conference Series*, 78 15
- [9] <http://code.macourmoyer.com/thin/>
- [10] <https://tools.ietf.org/html/rfc7292>
- [11] <https://wiki.nikhef.nl/grid/LCMAPS>
- [12] <http://www.egi.eu>
- [13] R. Moreno-Vozmediano, R. S. Monero, I. M. Llorente, IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures, *IEEE Computer*, vol. 45, pp. 65-72, Dec. 2012
- [14] <https://www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf>
- [15] K. Keahey, I. Foster, T. Freeman, X. Zhang, D. Galron, Virtual Workspaces In The Grid, *Europar 2005*, Lisbon, Portugal, Sep. 2005.
- [16] <http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=X.509>
- [17] S. Timm, K. Chadwick, G. Garzoglio, S. Y. Noh, Grids, virtualization, and Clouds at Fermilab, in *Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013)*, *Journal of Physics: Conference Series* 513 (2014). D. L. Groep and D. Bonacorsi, eds. IOP Publishing.
- [18] G. Garzoglio, J. Bester, K. Chadwick, D. Dykstra, D. Groep, J. Gu, T. Hesselroth et al. "Adoption of a SAML-XACML Profile for Authorization Interoperability across Grid Middleware in OSG and EGEE." In *Journal of Physics: Conference Series*, vol. 331, no. 6, p. 062011. IOP Publishing, 2011.
- [19] P. Mhashilkar, A. Tiaradani, B. Holzman, K. Larson, I. Sfiligoi, and M. Rynge, Cloud Bursting With GlideinWMS: Means to satisfy ever increasing needs for Scientific Workflows. In *Journal of Physics: Conference Series* 513 (2014). D. L. Groep and D. Bonacorsi, eds., IOP Publishing.