CHEP 2018
23RD INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS
9-13 July 2018
National Palace of Culture
Sofia, Bulgaria

THE ATLAS ACCESS MANAGER POLICY BROWSER: STATE-OF-THE-ART WEB TECHNOLOGIES FOR A RICH AND INTERACTIVE DATA VISUALIZATION EXPERIENCE

Introduction

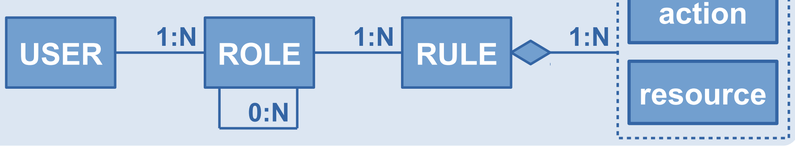
The ATLAS experiment comprises a significant number of hardware and software resources accessed and operated by many users. The Access Manager (AM) implements the resources access restriction to reasonable levels required for the successful running of the ATLAS experiment. Given the large number of ATLAS users, the AM only grants resources usage permission to users according to their current duties.

The service is designed on top of the Role Based Access Control (RBAC)[1] model. Every role describes a set of actions granted to users (rules). The rules allow login to a set of machines, to execute some commands with the security privileges of another user (via sudo), or to perform Trigger and Data Acquisition (TDAQ) system specific operations. The roles can be defined using inheritance. Every ATLAS user has a well-defined set of access privileges corresponding to a specific set of assigned and enabled roles. In total, there are several hundred roles and several thousand users.

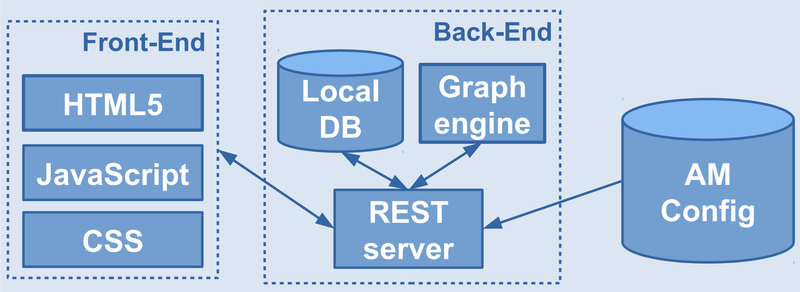
Given the size and the complexity of the system, a tool to browse and inspect the AM configuration is required. We present the deployment of a new visualization tool named Policy Browser. It is the primary tool for role administrators to inspect all the aspects of the AM configuration via a rich web-based interface.

RBAC

The RBAC model takes the access decision for an individual user on the basis of the roles the user has in the organization. A role includes rules defining permissions. A permission is the right to perform an action on a resource.



Back-end Overview

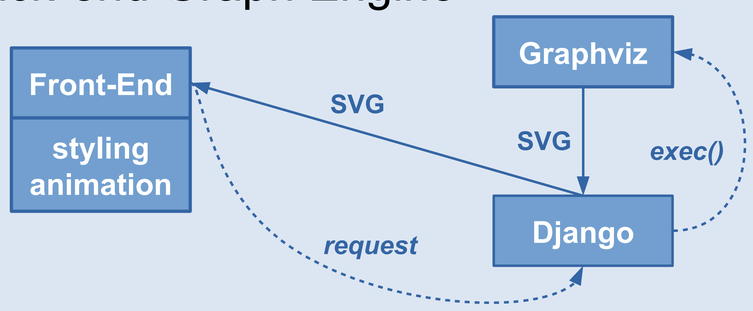


The Policy Browser is a web application exposing a REST[2] API. The server produces data (JSON) and graphs (SVG) and sends them to the client that displays them.

The server is using Django[3], a Python framework that follows the Model-View Controller (MVC)[4] architecture. The selection of Python was particularly interesting to do short prototyping iterations at the beginning of the project and the MVC architecture allowed to focus code development on the specific features of the project.

The client is using AngularJS[5], an open source JavaScript framework, in conjunction with Bootstrap[6], an open source toolkit for CSS development. It allowed producing a state-of-the-art GUI in a short time span.

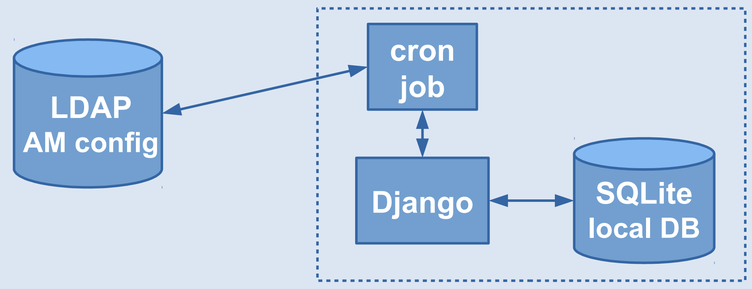
Back-end Graph Engine



The roles are organized into an inheritance hierarchy reflecting different levels of users expertise. For a given role, it is often necessary to visualize its “inherits from” and “inherited by” graphs.

There are various ways to draw graphs. Comparing them objectively requires defining appropriate metrics. In the case of a dependency graph (directed acyclic), a common approach is the layered graph drawing (also called *Sugiyama* drawing). Ideally no edge should go upward, and the goal is to minimize the number of edge crossing for readability. As for a large graph this can be a costly operation, the server relies on a dedicated graph engine, Graphviz[7], for all the graphs generation.

Back-end Storage and Data Update



The actual AM permissions used by the system are stored on LDAP server. Calculations of their interrelations for visualization require complex manipulations with its data.

The first in-memory prototype resulted in a complicated and hardly maintainable codebase. The second implementation leveraged a relational database (SQLite[8]). It allowed to make the code more readable and to benefit from the query optimizer for a fast response time. Django ships with an Object-Relational Mapping (ORM) that accelerates the initial design of the schema and simplifies data access. A cron job periodically checks if the permissions have changed, and updates the local database accordingly.

Front-end Overview

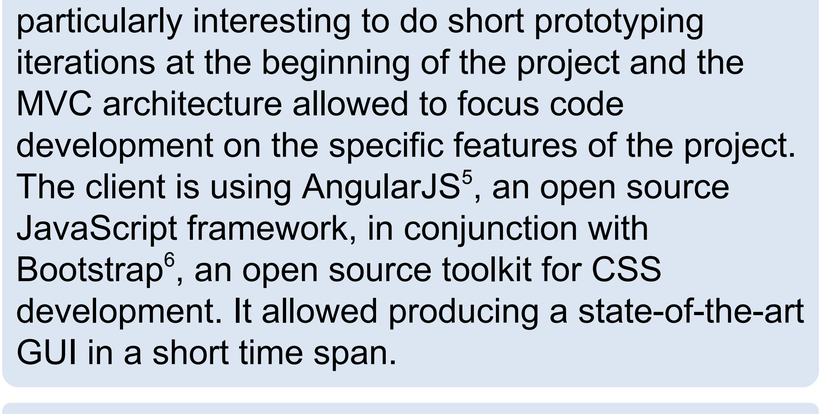
The web interface consists of data panels to display roles, users and permissions. A tree widget compacts thousands of objects by few resource categories and subsystems. For a specific object, we display the list of related authorization (e.g. for a given command it displays a list of roles that allows its execution and the list of authorized users). It also proposes text and category filters to further help finding the appropriate resource.

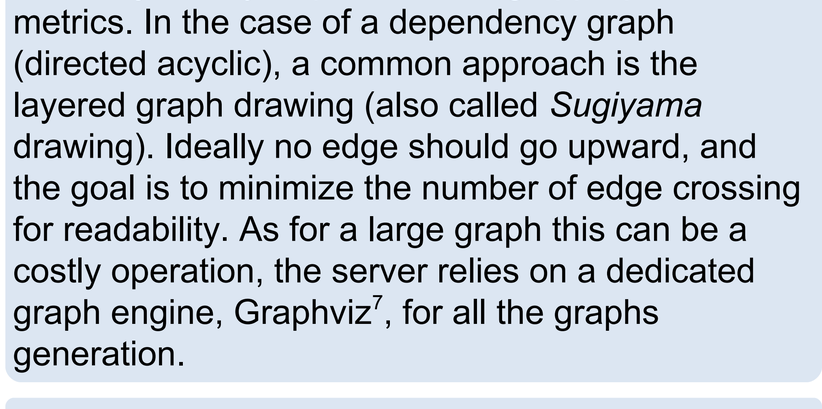
Front-end Graph Explorer

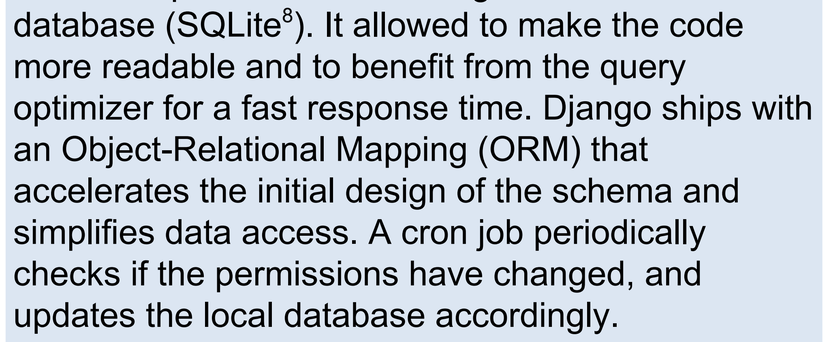
The Graph Explorer is a tool that allows displaying the inheritance hierarchy. It allows browsing large graphs comprising many layers and hundreds of roles with zoom and pan. It permits easy navigation from role to role with a graph morphing animation. Lateral filters are shown in order to highlight subgraph and an advanced search mode allows to fully customize the display.

Front-end Detail Graph

The role is the key element granting permissions. A user may have permission granted by several roles. Sometimes it is required to know not only which resource a user is accessing, but also via which role. This is where the detail graph is useful. Since often several paths allow access to a resource, the detail graph is an effective way to represent this information.







Conclusion

The application has been deployed to the ATLAS experiment site for production. It uses live data from the experiment. All the initial requirements have been implemented. Performance-wise, the graph generation is the critical part of the API. Benchmark results show that the current implementation requires an average of 70 ms per graph (CERN CentOS 7, i7-3770 CPU) which should not be perceptible by end users. The preliminary results from informal user feedback are generally very positive. They tend to indicate an intuitive layout and general ease of use.

References

1 RBAC https://csrc.nist.gov/Projects/Role-BasedAccess-Control
2 REST https://www.w3.org/2001/sw/wiki/REST
3 Django https://www.djangoproject.com
4 MVC https://dl.acm.org/citation.cfm?id=50757.50759
5 AngularJS https://angularjs.org
6 Bootstrap https://getbootstrap.com
7 Graphviz https://www.graphviz.org
8 Sqlite https://www.sqlite.org

Giuseppe Avolio[1], Sami Perrin[2], Igor Soloviev[3]
[1] CERN, [2] Ecole polytechnique Fédérale de Lausanne, [3] University of California, Irvine

ATLAS EXPERIMENT