NextLabs CloudAz

Help System

Revision 1

September 2018

CONFIDENTIALITY NOTICE

To provide feedback on this document, email the documentation team at techpubs@nextlabs.com.

TRADEMARKS

LICENSE AGREEMENT

Published in San Mateo, CA, by NextLabs, Inc.
www.nextlabs.com
info@nextlabs.com
CloudAz-Support@nextlabs.com
800.898.3065

Document version number: CC_CloudAzHelp_8.6.4.1_Rev1

Contents

REST API

Product Guide

QuickStart Guide

Using the QuickStart Guide

The QuickStart Guide explains how to use sample ABAC (Attribute Based Access Control) policies and configure a sample application to make authorization requests in NextLabs Cloud Authorization Service. This service is referred to as CloudAz. You access the QuickStart Guide from the *Help* section of the CloudAz console.

Topics in this section

QuickStart objectives

After completing the QuickStart Guide, you will:

- Understand the basics of ABAC (Attribute Based Access Control) and CloudAz
- Understand how a simple authorization policy is used in CloudAz
- Be able to perform a simple authorization using CloudAz

Before you begin

To use the QuickStart Guide example, you need to import sample ABAC policies into your CloudAz instance. The sample policies are for authorization in a typical Support ticket system. There are three authorization rules to be enforced by the Support ticket system:

- Allow support representatives to edit and reassign tickets that belong to their product area
- Allow users from the IT department to view all support tickets
- Deny access to security vulnerabilities if they are not created by or assigned to the user

Import the sample ABAC policies

To see how policies are used to control authorization, import the sample ABAC policies from the CloudAz Help page.

Procedure

1. Click the **Import sample policies** link on the in-product version of the QuickStart Guide. To access the in-product QuickStart Guide, click **Getting Started** under the Help icon in the upper right of the CloudAz console, then click the **QuickStart** link.
2. In the CloudAz console, click **Policies**. The imported policies appear on the *Policy Management* page.
3. Verify that the imported policies are in the *Deployed* state so that they are ready to be used in the QuickStart example. See Figure 1-1.

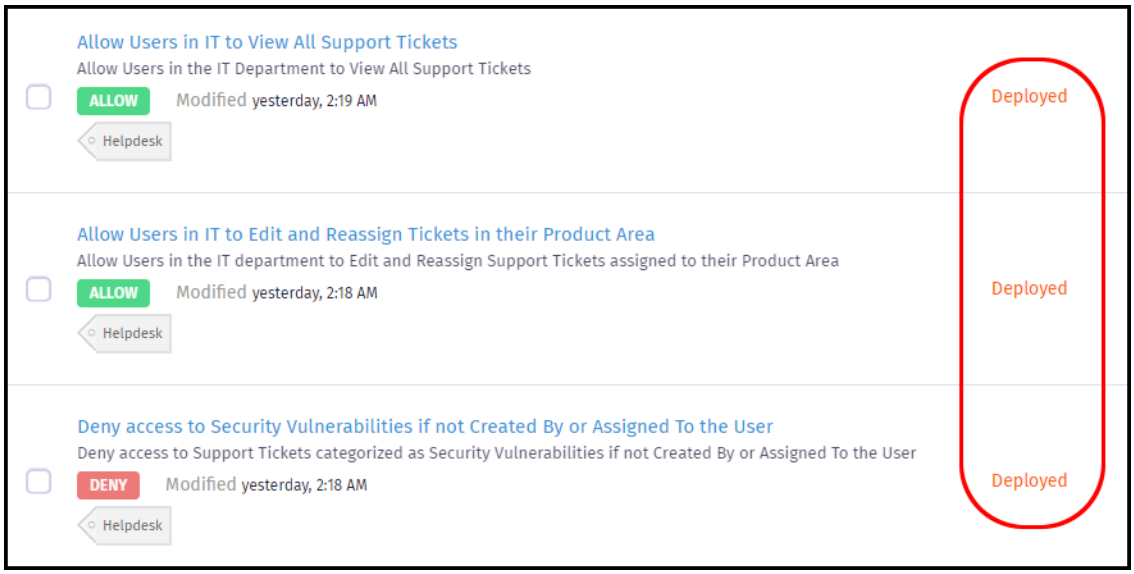


Figure 1-1: Imported policies in the Deployed state

Run the application and send an authorization request

After you have imported the sample policies, you can run one of the provided sample applications, which sends a set of authorization requests to the CloudAz REST API. CloudAz evaluates each authorization request against the imported sample policies; calculates a decision, allow, deny, or indeterminate; and returns the decision with an optional set of obligations. Obligations are additional instructions the sample application needs to perform before proceeding with the application logic. The sample application displays each set of request attributes and the response on-screen before proceeding to the next request.

Run the sample Java application and send an authorization request

After you have imported the sample policies, you can run the sample Java application and send an authorization request to the CloudAz REST API.

Procedure

1. Set up the Java SDK with JDK 7 or higher. For instructions, go to http://docs.oracle.com/javase/7/docs/webnotes/install/.
2. Download the sample Java application from https://s3-us-west-2.amazonaws.com/nxtlbsrelease/Platform_SAAS/openAZ-pep/Nextlabs-OpenAZ-PEP.zip.
3. Extract the files from the downloaded package.
4. In the extracted folder, navigate to: `java/sample_code/config`.

   Note: Files are extracted with the *Read-only* attribute set in their properties. To view or change file properties in Windows, right-click the file, then select *Properties*.
5. Change the properties of the following file to make it editable, then open it in a text editor: `openaz-pep.properties`.
6. Replace the following placeholders with the properties for your system and save the file:
   - `<CloudAz REST API host>`: The hostname of your CloudAz instance, which was provided in the *Welcome* email from NextLabs. For example: `nxlpmdemo-jpc.pm.nextlabs.solutions`.
   - `<client_id>`: The API Client ID. For example, `apiclient`.
   - `<client_secret>`: The API Client Secret.
7. Send the sample authorization request:

   a Open a Windows Command prompt or Linux terminal, and navigate to the following folder: `java/sample_code`.

   b Run one of the following scripts to send the request:

   - Windows: `cloudaz_request.bat`
   - Linux: `cloudaz_request.sh`

   The response appears in the console. To learn more about the response, see Runtime execution: Attributes, request, and response.

Run the sample JavaScript application and send an authorization request

After you have imported the sample policies, you can run the sample JavaScript application and send an authorization request to CloudAz REST API.

Procedure

1. Set up the JavaScript SDK. For instructions, go to https://nodejs.org/en/download/.

2 Verify that the JavaScript node version is 4.5 or higher, and the npm version is 3.10.6 or higher. To check the versions, use these commands:

```
node --version
npm --version
```

3 Download the sample JavaScript application from https://s3-us-west-2.amazonaws.com/nxtlbsrelease/Platform_SAAS/openAZ-pep/Nextlabs-OpenAZ-PEP.zip.

4 Extract the files from the downloaded package.

5 In the extracted folder, navigate to: `js/sample_code/config`.

Note: Files are extracted with the *Read-only* attribute set in their properties. To view or change file properties in Windows, right-click the file, then select *Properties*.

6 Change the properties of the following file to make it editable, then open it in a text editor: `openaz-pep.json`.

7 Replace the following placeholders with the properties for your system and save the file:

- `<CloudAz REST API host>`: The hostname of your CloudAz instance, which was provided in the *Welcome* email from NextLabs. For example: `nxlpmdemo-jpc.pm.nextlabs.solutions`.
- `<client_id>`: The API Client ID. For example, `apiclient`.
- `<client_secret>`: The API Client Secret.

8 Open a Windows Command prompt or Linux terminal, and navigate to the following folder: `js/sample_code`.

9 Send the sample authorization request by running one of the following scripts:

- Windows: `cloudaz_request.bat`
- Linux: `cloudaz_request.sh`

The response appears in the console. To learn more about the response, see Runtime execution: Attributes, request, and response.

About the sample application

The sample application is a generic Helpdesk Support ticket application included in the CloudAz QuickStart Guide for demonstration purposes. You can download and run this application to learn how the CloudAz authorization service uses ABAC policies to control access to resources and user interface elements in a typical application.

The sample application demonstrates a simple scenario where users access a Helpdesk application to view and manage support tickets. Users should only be able to view, edit, and reassign a subset of the support tickets in the Helpdesk system based on their product assignment, department, and issue category. The sample application code only illustrates the runtime

interaction between an application, such as a Helpdesk application, and the CloudAz system to determine what support tickets a user is allowed or not allowed access and does not include a user interface.

High-level flow and terminology

In the QuickStart example, the Helpdesk sample application issues REST-based authorization requests to the CloudAz authorization service, illustrating what happens when users attempt to view, edit, or reassign Support tickets using the Helpdesk application. The CloudAz policy engine, also known as the Policy Decision Point, or PDP for short, evaluates these authorization requests against the sample policies defined in the CloudAz system. The PDP returns authorization decisions that state whether access should be allowed or denied. These decisions are based on the attributes of Support tickets and the attributes of users, such as department and product area assignment.

Applications such as the Helpdesk sample application are known as Policy Enforcement Points, or PEPs for short. In a real-world setting, the Helpdesk application (PEP) would enforce authorization decisions returned from the CloudAz service, (PDP), by showing only the authorized set of tickets in the Helpdesk user interface, and controlling the operations users can perform as required by policies.

Runtime execution: Attributes, request, and response

After you've imported the sample authorization policies and downloaded the Helpdesk sample application, you can run the application to see how it interacts with the CloudAz service. Let's walk through the runtime interactions:

Note: Interactions are the same for both the Java and JavaScript sample applications.

1. The user performs an action in the Helpdesk sample application, such as viewing, editing, or reassigning a Support ticket.

2. The Helpdesk sample application intercepts the event, the user action, and prepares an authorization request by collecting user, resource, and environment attributes. The CloudAz policy engine uses these attributes to determine whether the user should be allowed to perform the action. Attributes in this example include:

   - A set of subject attributes representing the user. These include the user ID, department, roles, and the product area to which the user is assigned.
   - A set of resource attributes representing the Support ticket. These include the ticket ID, priority, and severity; the product affected by the ticket; the ticket category; the user assigned to the ticket; and the user who created the ticket.
   - Environment attributes. These include how the user authenticated to the Helpdesk sample application, either multi-factor or http-basic.

   The authorization request also includes the action (operation) the user is trying to perform on the Support ticket (the resource): view, edit, or reassign.

3 The Helpdesk sample application (PEP) submits the authorization request along with the user, resource, and environment attributes, and the requested action, to the CloudAz REST API over HTTPS.

4 The CloudAz service (PDP) receives and evaluates the request and calculates the policy decision.

5 The policy decision, Allow, Deny, or Indeterminate, is returned to the Helpdesk application.

Allow and Deny policy decisions are self-explanatory.

Indeterminate responses, however, indicate that an Allow or Deny decision could not be made, because the attributes and actions do not match any policies. Application PEPs must be designed to interpret Indeterminate authorizations. If application PEPs take a secure-by-default posture, Indeterminate responses are treated as Deny responses. This ensures that users are allowed to see and do only those things they are explicitly permitted to do according to policy, and nothing else.

In a real-world example, the application PEP would enforce the authorization decision by either allowing or denying the action. In this example, however, the Helpdesk sample application simply prints the policy decision received from the CloudAz service on the screen.

6 In addition to a policy decision, the response from the CloudAz service might also include an optional set of obligations. Obligations are instructions the application must process before enforcing the policy decision. Obligations might include:

- Messages to be displayed to users explaining why actions are denied.
- Instructions to record sensitive actions in a log, even when those actions are allowed.
- Email to be sent to managers when users perform specified actions.

More advanced obligations could include conditions to be appended to a query before it runs against the database. This ensures that users only view records that match the conditions in the obligation.

About sample authorization policies

The sample authorization policies, which are available for import, represent security, business, and regulatory requirements. For the Helpdesk sample application, the application owner, or the security and compliance officer, has decided that the following requirements must be met when users access Support tickets:

1 Users in the IT department are allowed to view all Support tickets.

2 Users in the IT department are only allowed to edit and reassign Support tickets for their own product area.

3 Users are not allowed to view, edit, or reassign Support tickets flagged as security vulnerabilities, unless the ticket is either assigned to, or created by, the user. Users should be notified of any policy violations with information on who is currently assigned to the ticket.

About sample authorization requests and decisions

The sample application issues four different authorization requests to represent the following use cases and scenarios:

1 Chris Webber, a user in the IT department, tries to view a Support ticket related to the SharePoint product area. For the sample code, see the .java or .js version of `AllowITDeptUsersToViewTickets.java`.

Authorization decision: The CloudAz authorization service evaluates the request and responds with an Allow decision because the request matches the *Allow Users in IT to View All Support Tickets* policy and requirement 1) defined in the previous section.

2 Chris Webber, a user in the IT department responsible for the Exchange email server, tries to edit a Support ticket related to issues with the Exchange server. For the sample code, see the .java or .js version of `AllowUsersToEditTktsAssignedToThem`.

Authorization Decision: The CloudAz authorization service evaluates the request and responds with an Allow decision because the request matches the *Allow Users in IT to Edit and Reassign Tickets in their Product Area* policy and requirement 2) defined in the previous section.

3 Chris Webber, a user in the IT department responsible for the Exchange email server, now tries to view a Support ticket categorized as a security issue. For the sample code, see the .java or .js version of `DenyUsersAccessToSecurityTkts`.

Authorization Decision: The CloudAz authorization service evaluates the request and responds with a Deny decision based on the *Deny access to Security Vulnerabilities if not Created By or Assigned To the User* policy and requirement 3) defined in About sample authorization policies.

If the CloudAz policy engine (PDP) finds Allow as well as Deny policies that match a request, the authorization decision is generally Deny, because Deny policies override Allow policies. In this example, Chris is denied access even though the Support ticket is for an issue with the Exchange server and she works in the IT department. The ticket is categorized as a security issue and not created by, or assigned to, Chris.

In addition to the policy decision, the response from the CloudAz service also contains an obligation the Helpdesk sample application must process as part of policy enforcement. In this example, the obligation includes a message that must be displayed to the user with the reason for denying access. The message also includes the user assigned to the ticket and the ticket ID.

4 Chris Webber, a user in the Marketing department, tries to view a Support ticket related to the Exchange server. For the sample code, see the .java or .js version of `NoMatchingPolicyFound`.

Authorization Decision: The CloudAz authorization service evaluates the request and responds with an Indeterminate response, because no policies in the system match Chris and users in the Marketing department.

As discussed earlier, the Helpdesk sample application must determine how to handle Indeterminate responses. A secure-by-default system would treat this as a Deny decision and not allow Chris to view the Support ticket.

For more information about authorization policies, requests, and CloudAz components such as PDPs and PEPs, see the CloudAz Overview section of the online Help.

Take it for a spin

It's time to get your feet wet. You can modify the sample code and policies to see how changes affect the policy decisions returned from the CloudAz service.

Note: When you make changes, you must run the sample application to see the results. See Run the application and send an authorization request.

Suggested changes:

1. Change the user's *department* attribute to *IT* in the .java or .js version of `NoMatchingPolicyFound` and run the sample application again. What was the policy decision this time? If it changed, why?

   Answer

   Instead of Indeterminate, the CloudAz service now returns an Allow policy decision as any user in the IT department is allowed to view Support tickets based on the policy, *Allow Users in IT to View All Support Tickets*.

2. Change the *created_by* resource attribute to *chris.webber* in the .java or .js version of `DenyUserAccessToSecurityTks` and run the sample application again. What was the policy decision this time? If it changed, why?

   Answer

   CloudAz now returns an Allow policy decision. Since the security-related Support ticket is now assigned to the user, the *Deny access to Security Vulnerabilities if not Created By or Assigned To the User* policy no longer matches the request. *Allow Users in IT to View All Support Tickets* is now the only policy matching the request.

3. Change the *product area assignment user* attribute to *SharePoint* in the .java or .js version of `AllowUsersToEditTktsAssignedToThem` and run the sample application again. What was the policy decision this time? If it changed, why?

   Answer

Instead of Allow, the CloudAz service now returns an Indeterminate policy response because there are no policies that match the request. Users in the IT department can edit tickets only if the users are assigned to the same product area as the ticket. There are no policies that describe what should happen if the user tries to edit a ticket assigned to a different product area than the user.

Now change the *category* resource attribute to *security* and see what happens when you run the sample application. The policy decision changes to Deny based on the policy, *Deny access to Security Vulnerabilities if not Created By or Assigned To the User*.

About the Policy Model

NextLabs CloudAz provides an easy to use, guided, policy authoring experience. Authorization policies are based on named subject and resource components that define a set of attribute conditions such as *Age > 21* or *Document Status = Public*. Components can be defined once and used in multiple policies. Any changes made to a component, once that component is activated, automatically take effect across all policies that reference the component.

The three sample policies in the QuickStart example are based on two resource components:

- **Security Vulnerabilities** which represents Support tickets categorized as security issues
- **Tickets in User's Product Area** which represents Support tickets that match the product area assignment of the user

The sample policies also reference a subject component named *IT Department* that defines the attributes and conditions representing users that belong to IT. Each component includes one or more conditions comparing resource attributes against either literal values or other attributes.

Components in turn are based on subject types and resource types defined in the CloudAz Policy Model. CloudAz includes a default User subject type in the Policy Model. The User subject type, which can be updated to include any user attributes that the policy system should be aware of, is available to administrators when they define the subject components used to grant or deny access. In the Helpdesk example, the User subject type has been extended to include user attributes representing the user ID, the department a user belongs to, the product area the user is responsible for, and the roles assigned to the user. A role is simply another User (subject type) attribute that you can base your policies on in an ABAC system such as CloudAz.

A resource type defines a set of metadata representing a category of protected objects in your application. Examples of resource types include Support tickets, customer records, and documents. Resource type metadata includes the resource attributes, actions, and obligations that are used by administrators, the CloudAz policy system, and the application (PEP) when authoring, evaluating, and enforcing policies.

The Helpdesk example includes a resource type named Support Tickets. This resource type defines the resource attributes (ticket ID, priority, severity, product area, issue category, assigned to, and created by), their datatypes, and the allowable values that are presented to policy administrators. These are passed at runtime to the CloudAz service by the Helpdesk sample application and used by the CloudAz PDP to evaluate authorization requests.

The Support Tickets resource type also includes the actions, such as view, edit, and reassign, to be controlled in the Helpdesk sample application. Whenever the Helpdesk sample application requests an authorization decision from the CloudAz service, it includes, as part of the authorization request, the action (operation) the user is trying to perform on a Support ticket.

Last but not least, the resource type also defines the name and parameters for any obligations to be defined in Support ticket policies and enforced in the Helpdesk sample application. At runtime the CloudAz policy engine returns an optional list of obligations with the final policy decision (based on the effective policies that match the authorization request at runtime) to further affect policy enforcement in the Helpdesk sample application. In this example the Support Tickets resource type includes the definition and structure of a Policy Violation Notification obligation that includes three parameters (*ticket_id*, *message*, and *assigned_to*). CloudAz administrators who author policies related to Support tickets can, as part of the policy, define a message that should be presented to the end user when a given policy is enforced. This is the obligation included in the policy example, *Deny access to Security Vulnerabilities if not Created By or Assigned To the User*. The *ticket_id* and *assigned_to* attributes included in the authorization request are also returned as part of the obligation so that the application can include these in the message to the user.

Creating components and policies

To demonstrate how resource types drive the policy and component authoring experience, you can create a new resource component and policy that allows users to view any Support tickets they have created.

Procedure

1 Log in to the CloudAz service using the URL and credentials provided in the *Welcome* email from NextLabs.

2 Create a new resource component:

a On the left navigation bar, select **Components > Resource**, then select Add Component.

b In the *Name* field, type `My Own Support Tickets`.

c Select the Support Tickets resource type.

d Add the following condition: `Created By = $(user.user_id)`

This condition compares the resource attribute *created_by* against the *user_id* subject attribute. Note that the left side *Name* field in the condition editor lists the attributes defined in the Support Tickets resource type. An upcoming build of the CloudAz service will also enable you to select user and resource attributes from a list in the right-hand *Value* field in addition to selecting them on the component form. Note that multiple conditions in a component are treated as AND statements by the CloudAz PDP.

e Save and deploy the component.

3 Create a new policy:

a On the left navigation bar, click **Policies**, then click **Add Policy**.

b In the *Name* field, type `Allow users to view their own Support tickets.`

c Define the policy as an Allow policy.

Important: Do not add any subject components to the policy. An empty subject component field means that the policy applies to any and all users.

d Add the *My Own Support Tickets* resource component you created earlier to the policy.

After adding the resource component, note that the bottom of the CloudAz policy was updated with the list of obligations defined in the Support Tickets resource type. In a real-world scenario, the Policy Violation Message obligation defined here probably applies to Deny policies only, but can be enabled if desired.

e Add the *View* action component to the policy. Note that the actions in the list are restricted based on the previously selected resource components and the corresponding resource type(s).

f Save and deploy the policy.

4 On the left navigation bar, click **Dashboard** and refresh the screen until the outer circle in the top left *System Configuration Status* widget turns blue.

Blue indicates that the new policy and component has been activated and distributed to the CloudAz PDPs and is available for runtime policy evaluation. The PDPs, which are running on a separate set of distributed servers, ping the CloudAz policy server (Policy Administration Point, or PAP for short) at regular intervals, typically once a minute, to obtain new or updated policies and components.

Voila! Users are now allowed to view their own Support tickets in the Helpdesk sample application. You can demonstrate this by modifying the user and resource attributes in one of the included sample applications.

Go wild!

Now that you understand how applications (PEPs) interact with the CloudAz service, you can integrate your own application with CloudAz and take advantage of all that CloudAz has to offer.

For more information about CloudAz components, see Overview of CloudAz. For more information about ABAC, see Overview of ABAC.

ABAC Concepts

Overview of ABAC

This section describes ABAC (Attribute Based Access Control) and explains how ABAC differs from other access control systems such as RBAC (Role Based Access Control) and ACL (Access Control Lists).

Topics in this section

Introduction to ABAC

ABAC (Attribute Based Access Control) is a logical access control methodology that uses attributes, or properties, of users, resources, and the environment to determine whether to allow or deny access to assets. Because they are based on attributes, ABAC policies continue to correctly control access to assets even when users change departments, projects, locations, or security clearance levels, and when documents and other resources are reclassified or modified.

Examples of attributes include:

- Department
- Citizenship
- Credit score
- Territory assignment
- Purchase order amount
- Information about health records
- Time of day
- Authentication level
- Device status
- Location
- Context or environment

Attributes can be imported from various enterprise information systems such as Active Directory servers, HR (Human Resources) systems, and any existing attribute store.

How ABAC policies are evaluated and maintained

ABAC policies are dynamically evaluated at runtime. When a subject requests access, the ABAC engine makes an access control decision based on the assigned attributes of the requester, the

assigned attributes of the object, the environment conditions, and a set of policies that are specified in terms of those attributes and conditions.

ABAC policies primarily have four main components:

- **The resource**: The logical entity to be secured from unauthorized access.
- **The subject**: An entity, usually a human, requesting access to the resource.
- **The action**: The operation the subject is trying to perform on the resource. For example, view, edit, or approve.
- **The policy effect**: The action to be take when the conditions match or do not match. The policy effect is either *allow* (sometimes called *permit*) or *deny*.

Policies require no direct reference to potentially numerous users and objects, and users and objects can be provisioned without reference to policies.

Additionally, if policy designers need to change policies to meet changing regulatory requirements, they can do so easily without going through lengthy development, test, and quality assurance cycles. The ABAC work flow is illustrated in Figure 2-1.

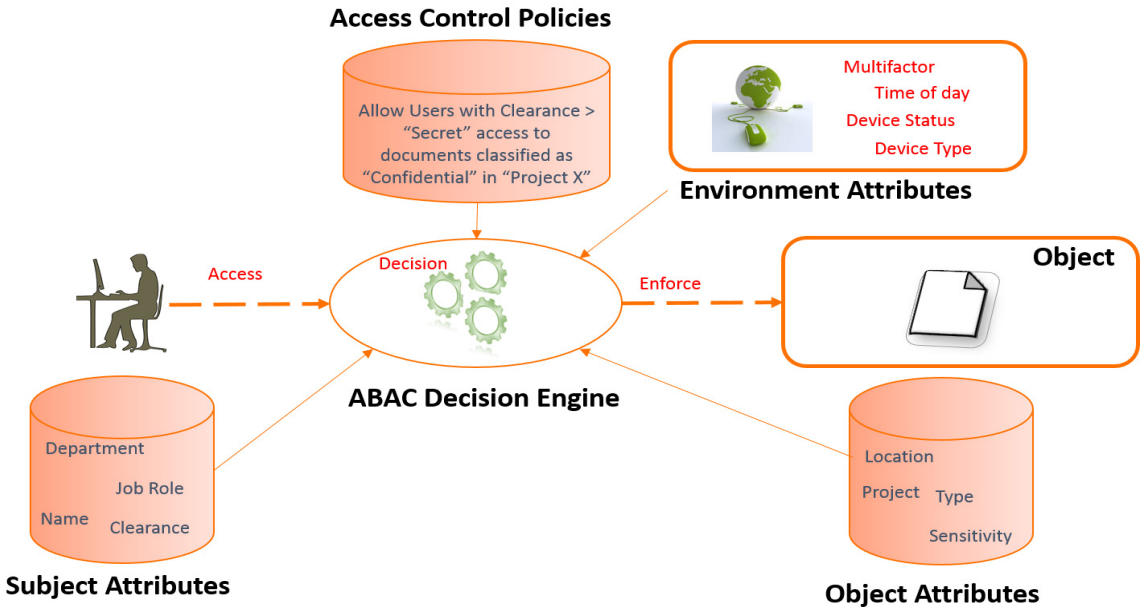


Figure 2-1: ABAC overview

Basic ABAC architecture

There are multiple frameworks that can be used to build an ABAC system. One of these uses XACML (eXtensible Access Control Markup Language). XACML supports a distributed architecture with various key components that are required to author, evaluate, and enforce policies.

The components and their roles in this framework are:

- **Policy Administration Point (PAP)**: The component that enables users to author and manage policies. PAP can also provide additional features such as reporting and activity monitoring to support audit capabilities.
- **Policy Decision Point (PDP)**: The component that evaluates applicable policies and renders an authorization decision.
- **Policy Enforcement Point (PEP)**: The component that performs access control by sending decision requests to the PDP and enforcing authorization decisions.
- **Policy Information Point (PIP)**: The component that is the source of user, resource, and environment attributes. A PEP or PDP interacts with this component to obtain attributes. In an enterprise deployment, a typical PIP would be an LDAP server, HR database, or other data repository with license keys.

These components are illustrated in Figure 2-2. For more information about XACML, go to https://www.oasis-open.org/committees/xacml/.

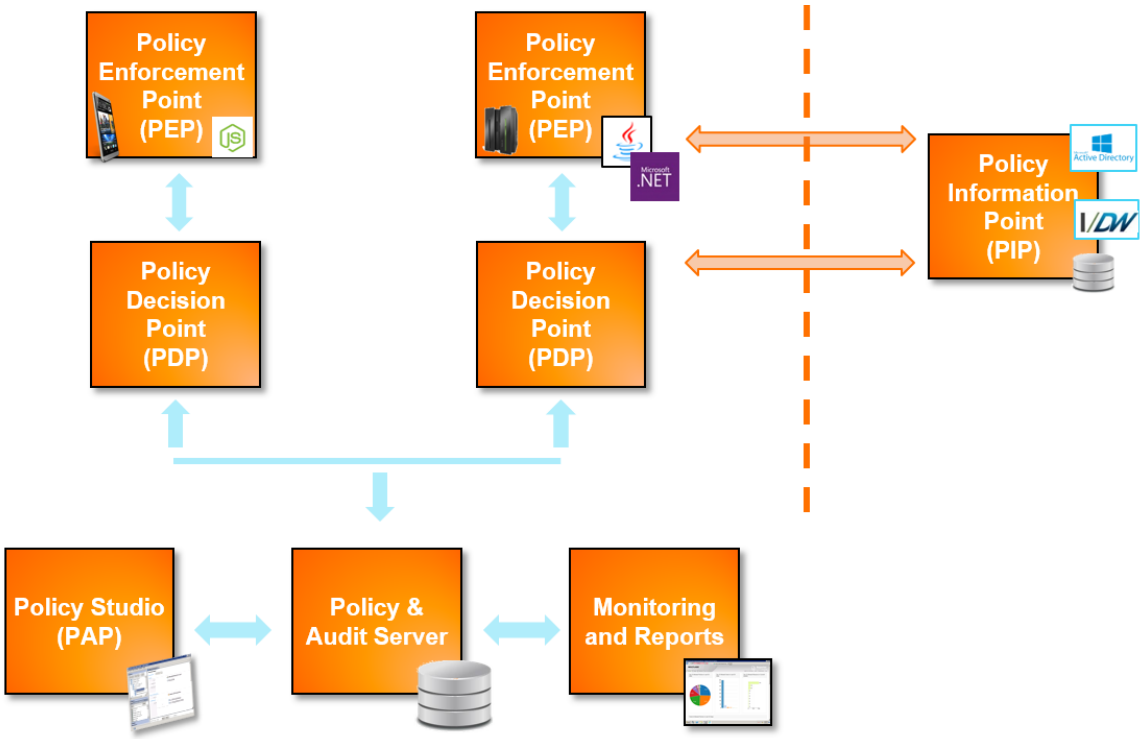


Figure 2-2: How ABAC policies are implemented

Comparing ABAC, RBAC, and ACL models

The key difference between ABAC and its predecessor access control models, including Role Based Access Control (RBAC) and Access Control Lists (ACL), is the ability to express complex

business rules as logical expressions using attributes. This ability makes ABAC a dynamic, context-aware, and highly scalable solution for access control.

By contrast, RBAC and ACL are static models that do not automatically adapt to changes in user status or data classification, and they do not handle environment-related conditions such as multi-factor authentication. Further, they provide no central visibility or control, and maintaining them can require considerable administrative resources.

ABAC implementations typically require only a small set of policies to be created and maintained based on business requirements, whereas RBAC and ACL are more administratively intensive to maintain. Further, because roles and permissions must be applied manually, RBAC and ACL models are more prone to error than ABAC models.

About RBAC models

RBAC (Role Based Access Control) models control access to resources using roles. Typically, roles correspond to functional responsibilities in organizations. For example, roles in an IT department might include Network Administrators, Security Analysts, and Helpdesk or Support Technicians. In a Human Resources department, roles might include Recruiters, Compensation Analysts, and Benefits Managers. To use RBAC, these roles need to be created, along with their permissions, and properly assigned to users in the enterprise. Further, each role might need to be created for each region or customer territory, resulting in role explosion as shown in *Figure 2-3*.

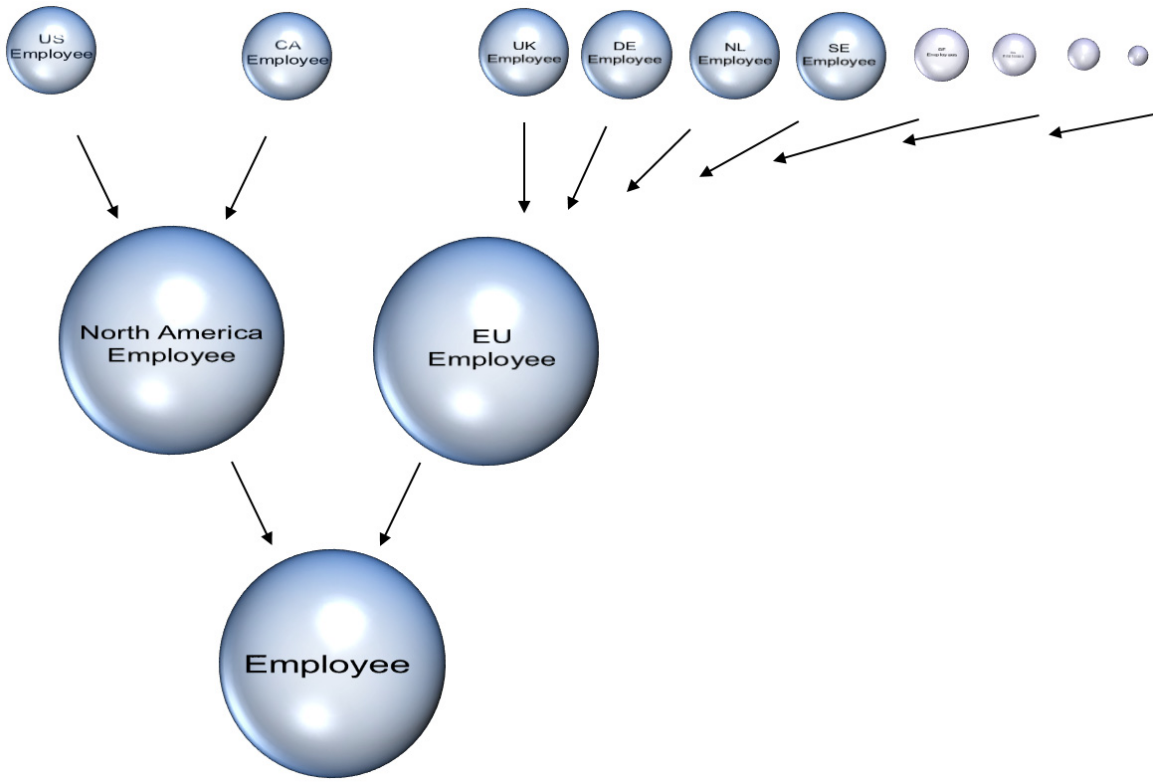


Figure 2-3: Role explosion

Consider a large organization with global operations and a business need to limit access to resources based on region and functional responsibility. To use RBAC to protect resources, the organization would need to create roles for employees in every region, assign the correct permissions to those roles, and assign the appropriate roles to users. The organization could easily require thousands of roles to cover each region and operational area. Further, maintaining those roles, and ensuring that user roles are updated when functional responsibilities or resource access requirements change, requires significant administrative resources.

For more information about RBAC, see http://csrc.nist.gov/groups/SNS/rbac/.

About ACL models

ACL (Access Control List) is the traditional access control model used for file systems, firewalls, databases, routers, and content management systems. ACLs specify the permissions that are either granted or denied to users and groups of users. For example, an ACL might grant a file owner, such as a System Administrator, the ability to read, write, and execute actions on a file. For analysts and technicians, the ACL might grant read-only permissions, and it might prevent users outside the IT department from having any access to the file.

Like RBAC, ACL is a static model. ACLs must be continually updated when functional responsibilities and resource access requirements change. Further, each file and directory has ACLs, and because ACLs have no central management, remembering permission settings and keeping permissions up-to-date requires significant administrative vigilance.

For more information about ACLs, see https://msdn.microsoft.com/en-us/library/windows/desktop/aa374872(v=vs.85).aspx.

Comparing ABAC, RBAC, and ACL implementations

This section compares how ABAC, RBAC, and ACL models can be used to enforce a business rule.

Business rule

Each access control model must enforce the following rule:

> Allow US citizens to view and edit Secret and Top Secret documents if the user's security clearance is higher than or equal to the sensitivity classification of the document.

ABAC implementation

This section assumes that the documents covered by the rule have the appropriate security Classification Level applied.

To enforce the Business rule using ABAC in CloudAz, a policy designer writes a policy that allows View and Edit access to documents if the users' Citizenship attribute equals USA and their Security Clearance attribute is higher than or equal to the document's Classification Level attribute.

When document Classification Levels and user Security Clearances change, CloudAz continues to enforce the policy dynamically at run time and provides the appropriate level of access without requiring modification to the policy or its components.

RBAC implementation

To enforce the Business rule using RBAC:

1. In a role management system, create the required roles. For example: Citizenship and security for each country where the rule needs to be enforced. For example, USA: Top Secret, Secret, Confidential and so on. If you have 5 levels of security and 10 countries, you would need 50 roles.
2. Assign roles to users using information from a Human Resources system.
3. For each role, add the appropriate permissions for viewing and editing documents.
4. Update document permissions whenever new documents are added.
5. Update role assignments whenever user Citizenship or Security Clearance changes. Keeping these assignments up to date requires ongoing administrative diligence, which is less efficient than ABAC.

ACL implementation

To enforce the Business rule using ACLs:

1. For each document that matches the business rule, define ACL permissions in the document properties. This requires that you know the Citizenship and Security Clearance level of each user or group of users and the classification level of each document. Figure 2-4 shows an example of the ACL permissions that need to be defined for files.
2. Add exceptions to the business rule to document properties as needed.
3. Update ACL permissions whenever users are added and whenever user citizenship or security classifications change.
4. Update ACL permissions whenever documents are added, and whenever document classifications change.

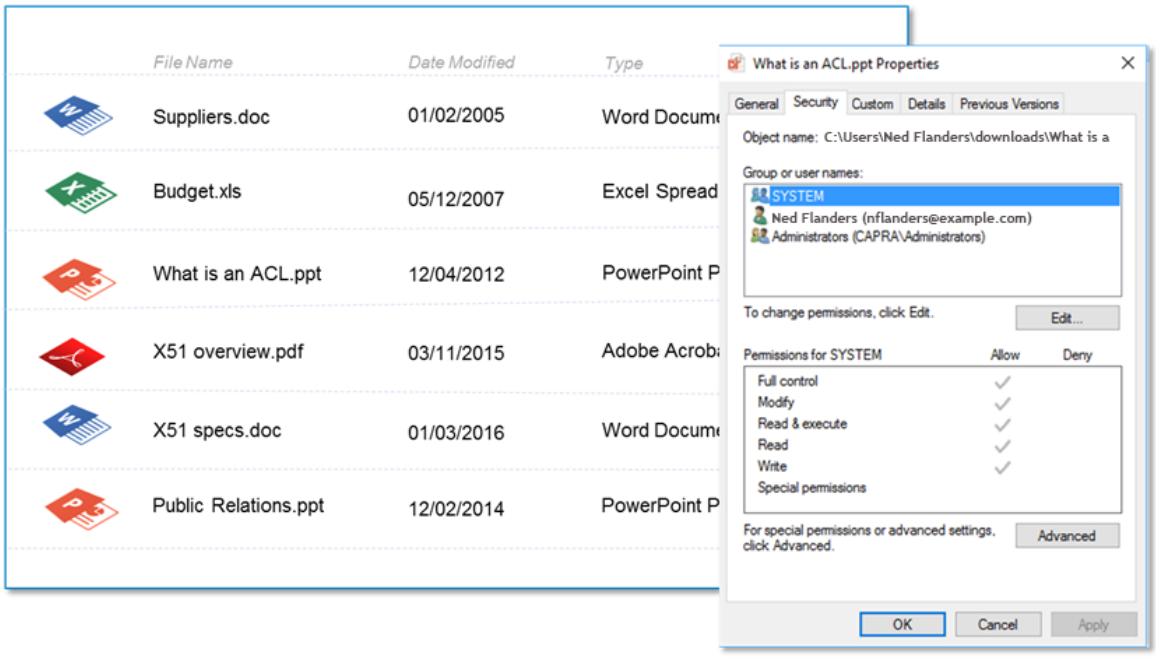


Figure 2-4: Access Control List example

ABAC benefits

ABAC has these benefits:

- Centralized policy management
  - **Centralized visibility and control**: All data and application access policies can be created and managed in one platform. This makes it possible to audit information related to user activity and data access, which enables applications using ABAC to achieve a high level of regulatory compliance.
  - **Externalize authorization policies from application logic**: ABAC has a simple API-based integration that hides the computational complexity of authorization from the application where security is enforced. In other words, policies are external and separated from application logic, and changes in regulatory requirements, business requirements, and security policies have no impact on applications. Further, because authorization policies are not embedded as code in the application logic, they can be managed outside of applications. This enables policy designers to centrally author, review, and update authorization policies whenever requirements or market conditions change. This eliminates the requirement to review code with auditors, and provides central visibility and control over access across applications.
  - **Separate policy lifecycle from application lifecycle**: The ABAC policy lifecycle is independent of the software lifecycle. As a result, policies can be changed and implemented immediately in response to changing requirements or market conditions. If the authori-

zation logic were embedded in the application code, application developers would need to modify the application code, go through lengthy development and test cycles, and eventually find a window of time where the application could be taken down and upgraded with the modified application logic. With policies managed externally from the application, changes can be made effective immediately in the production environment if required, without any application downtime or lengthy development and test cycles. To verify that policy changes are correct before they are implemented in production systems, short test cycles can be conducted in development environments, avoiding the need for application downtime. This enables organizations to get applications to market and react to changing conditions much faster, providing dramatic cost and time savings, improving time to market, and even providing an edge over competitors.

    - **Policies can span multiple applications**: Because policies are managed outside the applications, the same authorization policy can be enforced in multiple applications. In other words, a single set of policies can be enforced consistently across applications, providing central visibility and control over access. For example, you can define a single policy to secure a document that moves between various applications such as email, content management, or business process applications, and enforce the same policy in all applications.
- **Dynamic authorization**: Authorization policies are dynamically evaluated at run time to determine whether user actions should be allowed or denied. Policies are based on user attributes, such as citizenship, security clearance, department, and roles; resource attributes, such as data classifications, content, and transaction details; and environment attributes, such as time of day, location, authentication scheme, and device type. Administrators do not need to maintain and keep track of role and permission assignments when users move between departments, projects, or locations, or when their security clearance level changes.
- **Dynamic context**: Authorization policies can specify that documents can be accessed only during specified hours or from specified locations.
- Improved compliance and business agility
    - **Compliance with regulatory requirements**: ABAC policies are centrally managed, and they provide visibility and control across many applications and data. This makes it possible to audit information related to user activity and data access, which enables applications using ABAC to achieve a high level of regulatory compliance.
    - **Business agility**: Authorization policies are managed externally from the protected application so they can be modified without requiring code changes or application downtime. This enables organizations to react quickly to changing business or regulatory requirements, which greatly increases agility and flexibility.
- Reduced operational costs
    - **Reduced software maintenance costs:** In a classic authorization model, authorization checks are built in to application logic. When requirements or market conditions change, application code has to be modified according to the software development lifecycle, increasing overhead and maintenance costs. In an ABAC system, the cost of maintenance is limited to changing and testing the policy itself.
    - **Administration costs**: Using a central policy repository and central audit repository for multiple applications reduces the administrative cost of maintaining multiple access control and reporting systems.
- Secure sensitive applications, services, and data

  - **Control over user access**: ABAC enables organizations to control who can perform a given set of actions under a given set of conditions. Use cases include who can access applications, the circumstances under which applications can be accessed, the parts of an application a given user can access, the business transactions a user is allowed to submit, the data users can see, and the actions users can perform with that data.
  - **Control over applications and systems**: ABAC can be used to protect single-page web applications, micro-services, mobile apps, and any other type of application or system. Applications integrate with the ABAC system using standards-based REST APIs or SDKs for a variety of programming languages.
- **Rich Policy Model**: ABAC policies are based on rules, logical expressions, and attributes that read like natural language statements and can be authored by business users without help from IT. This provides great flexibility in expressing complex business requirements in simple and easy to understand authorization policies.

Additional resources

For more information about ABAC concepts, see the National Institute of Standards and Technology (NIST) websites:

- NIST overview of ABAC:
  http://csrc.nist.gov/projects/abac/
- NIST guide to implementing ABAC:
  http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf
- NCCOE building blocks for ABAC:
  https://nccoe.nist.gov/projects/building_blocks/attribute_based_access_control

CloudAz Concepts

Overview of CloudAz

NextLabs CloudAz is the industry's first service that provides ABAC (Attribute Based Access Control) and Dynamic Authorization in the cloud. The service is available through a variety of subscription licensing options that are tailored to development, test, and highly available production deployments.

Topics in this section

CloudAz platform

CloudAz is based on standards such as XACML (eXtensible Access Control Markup Language), ABAC, and OpenAz and provides a platform for authorization. CloudAz comes with a highly scalable policy evaluation engine designed for securing mission-critical applications that run in the cloud, on-premise, or on mobile devices. Powered by proven NextLabs XACML-based Dynamic Authorization, CloudAz provides a central platform for policy management, audit, and activity monitoring.

CloudAz enables organizations to control:

- The circumstances under which users can access applications
- The parts of applications users can access
- The business transactions users are allowed to submit
- The data users can see
- The actions users can perform with data

CloudAz features

CloudAz features include:

- **Simple policy authoring**: CloudAz provides modern, easy-to-use, web-based user interfaces for managing policies. These interfaces give policy designers complete visibility and control over access to applications and data. Policies are described in natural language statements that business users can define without requiring assistance from developers or IT administrators.

- **Dynamic policy evaluation**: Authorization policies managed through NextLabs CloudAz are dynamically evaluated at runtime. With ABAC, administrators no longer need to maintain and track role and permission assignments when users move between departments, projects, or locations; when user security clearances change; or when documents and other resources are reclassified or modified. Authorization policies are dynamically evaluated at runtime and access is granted or denied based on information and attributes related to the user, the resources, and the environment.
- **Scalability**: CloudAz is based on a proven fully distributed architecture that scales to meet the most complex and demanding requirements of mission-critical Internet-scale applications. The system runs on data centers in the cloud for high availability.
- **Integration through REST APIs and client SDKs**: CloudAz provides standards-based REST APIs hosted in the cloud to quickly and easily integrate, protect, and control access to applications, services, and data. NextLabs CloudAz can help secure single-page web applications, micro-services, mobile apps, and other types of applications, regardless of where they are deployed. Application developers who interact with the APIs simply enforce authorization decisions returned from CloudAz in their applications; developers do not need to worry about how those decisions were made or the criteria (policies) on which decisions were based. Additionally, CloudAz provides standards-based client SDKs in a variety of programming languages to meet development needs.
- **Activity monitoring and auditing**: CloudAz tracks and stores user activity and data access across applications and services in a central audit repository. NextLabs CloudAz provides insight into user behavior through dashboards, reports, and automated monitoring facilities.

CloudAz functional components

CloudAz consists of two main components: The PAP (Policy Administration Point) server and the PDP (Policy Decision Point):

- **PAP server:** Also referred to as the CloudAz server or the Control Center server. This server consists of these key components:
  - **Policy management service**: The service that provides the interface used to author and manage policies.
  - **ICENet**: A CloudAz server component that communicates with the Cloud PDP to send policies and receive user activity records.
  - **Reports, Monitoring and Alerts**: The interface used to view and track user activity.
  - **Delegated Administration:** The interface used to control user access to the CloudAz console.
  - **Database repositories**: Used to store policies and audit records.
- **PDP**: The entity that evaluates applicable policies and computes authorization decisions.

For an illustration of CloudAz components and architecture, see Figure 3-1.

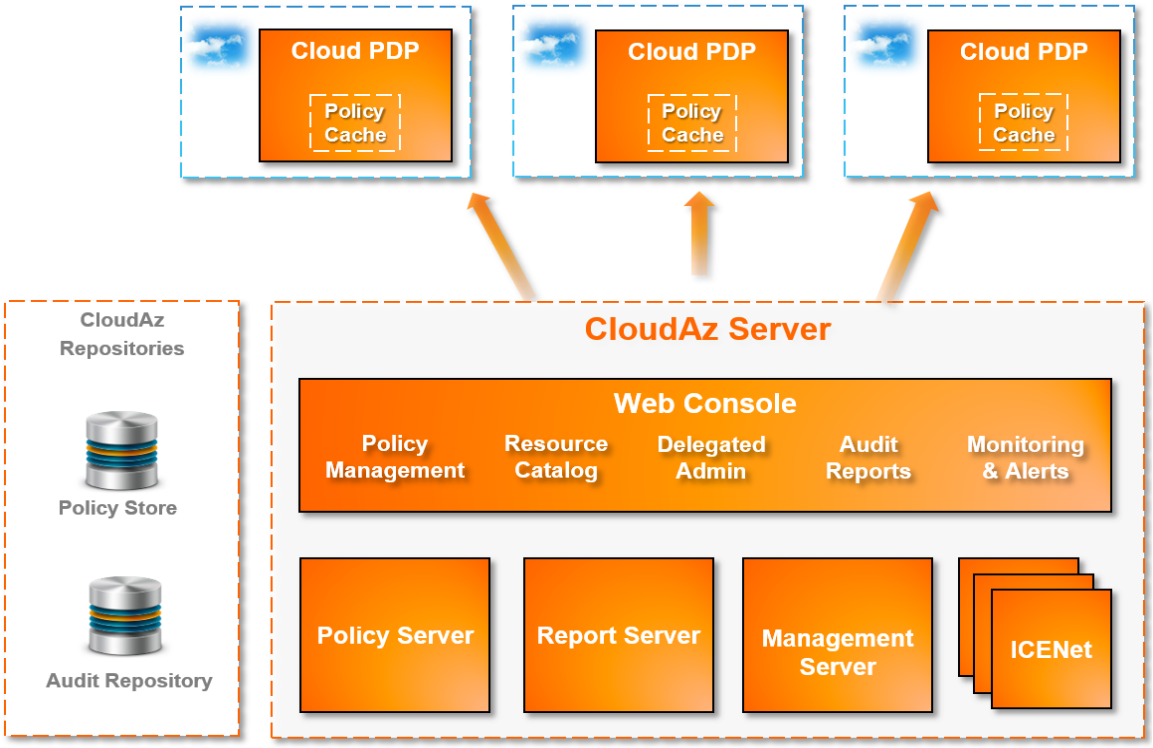


Figure 3-1: CloudAz component overview

CloudAz server

The CloudAz server provides the core functionality of CloudAz, including centralized policy management with audit and activity monitoring. In accordance with XACML standards, the CloudAz server acts as a PAP (Policy Administration Point). In addition to the policy management activities, the CloudAz server also provides a centralized view of user activity, data access reports, and configurable monitors and alerts to notify administrators about any out-of-the-ordinary activities. For details on XACML protocol and terminology, go to https://www.oasis-open.org/committees/xacml/.

Cloud PDP

CloudAz comes with a preconfigured set of PDPs (Policy Decision Points), which supports standards-based interaction REST APIs to make authorization requests. The PDPs evaluate authorization policies at runtime against incoming requests, and return authorization decisions to the PEP (Policy Enforcement Point). The REST APIs are based on the XACML 3.0 REST and JSON profiles. For high availability and scalability, the Cloud PDP is front-ended by a load balancer. The CloudAz PDP scales to meet any throughput requirements and can be adjusted based on the subscription level purchased.

The REST API supports single and bulk requests, and it handles XML and JSON payloads. To simplify the developer experience, CloudAz also provides client libraries in multiple programming

languages. Any HTTPS capable application, regardless of programming language or environment—single page apps (client-side JavaScript), mobile apps, and mainframes—can work with Cloud PDPs.

In addition to REST APIs, CloudAz supports a set of SDKs that are based on OpenAz standards and used to interact with Cloud PDPs. Using these SDKs, developers can build PEPs for mobile or web applications. See SDK and OpenAz Client Libraries. The SDKs internally convert the native payload into XML and submit the request via HTTPS as shown in Figure 3-2.

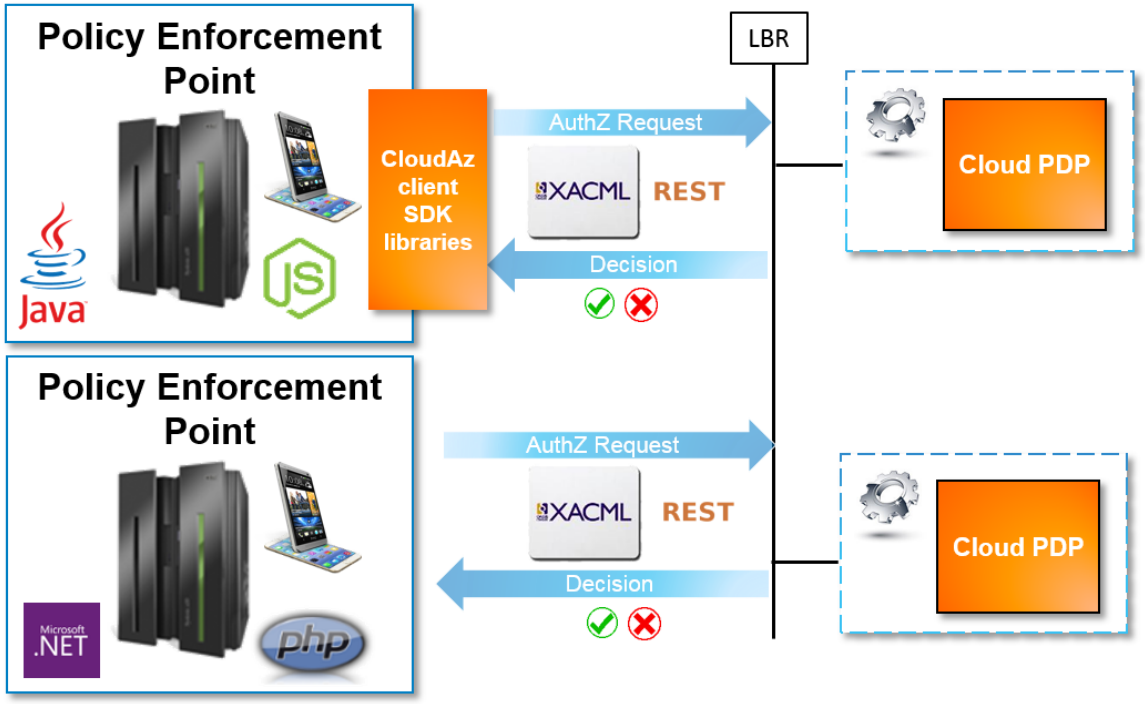


Figure 3-2: CloudAz architecture

Reports

To complete the authorization request lifecycle, CloudAz has a built-in reporting solution. This reporting solution is a centralized audit repository of all the user activity that the PDP captures. This includes request details such as the user attributes, resource attributes, and environment attributes used to evaluate the request. These details provide a comprehensive view for auditors and security administrators to understand why a user was authorized to access a resource or why access was denied. Additionally, CloudAz administrators can set up alerts based on logical rules, so that they can be notified of any suspicious activity. See About Reporter.

About the Policy Model

The Policy Model is the foundation of authorization policies, and it consists of subject types and resource types. A subject type is a metadata representation of the actor who is accessing an application and data in that application. In this version there is a single predefined subject type, which is *User*.

Each subject type is associated with a list of attributes, and each attribute is described by a data type and supported operators. For example, if a user is accessing a CRM (Customer Relationship Management) application as an employee, the user model could have these attributes:

- Subject Type: Employee (User)
  - Attributes:
    - First Name (String)
    - Last Name (String)
    - Email (String)
    - Department (String)
    - Job Role (List: Support Representative, Support Manager, and so on)
    - Assigned Product (String)
    - Citizenship (String)
    - Age (Number)
    - DoB (Date)

A resource type is a metadata representation of an object that is being secured. This metadata identifies the attributes that define the resource type and the actions that can be performed on the resource type. Resource types are then used to guide the component author and policy author experience. Examples of resource types include documents, customer records, health records, and bank accounts. A document resource type could include the following metadata:

- Resource Type: Document
  - Attributes:
    - Document Name (String)
    - Location (String)
    - Size (Integer)
    - Type (List: PDF, MS-WORD, TXT)
    - Project (String)
    - Sensitivity (String)
  - Actions:
    - View
    - Edit
    - Print
    - Download
    - Save
    - Email

- Obligations:
  - Encrypt Document (users can email documents, but have to encrypt them)

Figure 3-3 shows a UML (Unified Modeling Language) diagram of the data model. Primary keys are indicated by **#**. Required columns are indicated by *****. Optional items are indicated by **o**.

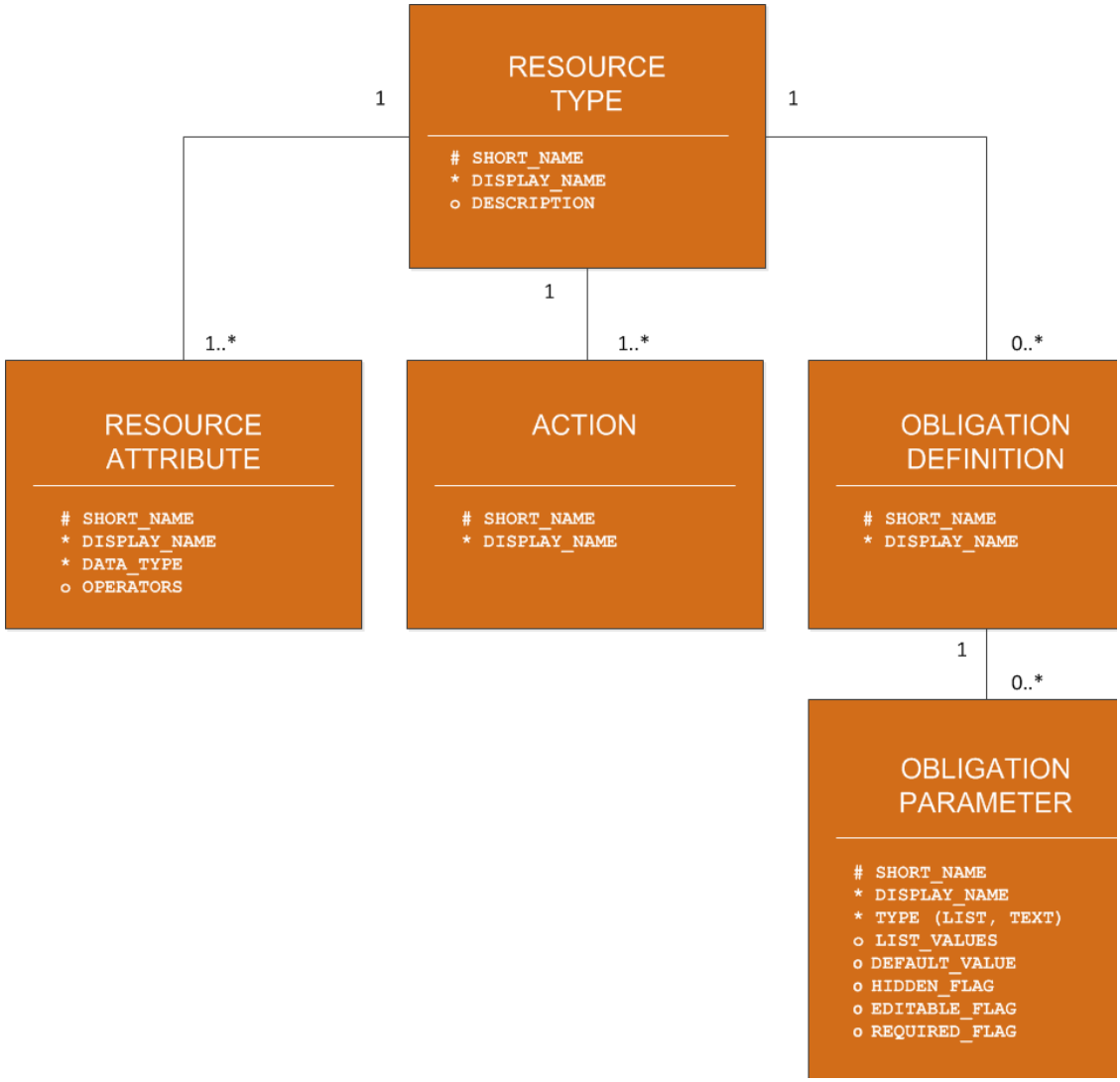


Figure 3-3: Resource types

About the CloudAz console

The web-based console of CloudAz enables administrators to manage policies, users, system configuration, and reports from a single interface. Console components are described in Table 3.1.

Table 3.1: CloudAz console overview

Section	Description
Dashboard	Summary information related to CloudAz components.The data displayed in all panes of the dashboard is refreshed from the Activity Journal each time you refresh the Dashboard page. This means that data is updated on demand. If a pane shows a statistic for the past week, that reflects not the last seven whole calendar days, but the last seven 24-hour periods starting from the top of the current hour.
Policies	Used to create and manage policies, which are the rules that govern information use. Constructing policies refers to the process of building information control policies in CloudAz. Policies use components as building blocks to represent rules that control information access and use in your organization. Deploying policies means the distribution of new or modified policies and policy components to the appropriate enforcement points on desktop PCs, laptops, and file servers throughout the organization. This means you can create, review, and refine policies as long as you like, but they are not enforced until they are deployed and the appropriate enforcers are configured.
Components	Objects used to construct policies. Components can be thought of as the parts of speech used to construct policy statements. For example, “noun” policy components might include *All employees in the human resources department* or *Any file with an .xls extension*, and “verb” components might include *Copy*, *Print*, and *Rename File*.
Policy Model	An abstract model that represents the various parts of the enterprise environment, such as users, applications, documents, and desktop PCs. The actual work of modeling consists of defining *policy components*. A component is a named definition that represents a category or class of entities, such as users, data resources, or applications; or of actions, such as Open or Copy.
Reports	The web-based application for creating reports and monitoring and tracking user activity, and it is the front end of the Report server.
Administration	Used to manage access to the CloudAz console and manage user accounts.
Delegation Policies	Policies that are used to manage access to the CloudAz interfaces as well as policies, and policy objects. Delegated Administration policies use rules based on user attributes and conditions. This mirrors the way policies are used to control access to an organization’s resources and eliminates the need to create and apply roles to users or groups of users. See Adding and editing delegation policies.
Users	Accounts that can be used to log in to the CloudAz console. See Managing user accounts.
Tools > Policy Tester	A utility used to test policies within the CloudAz console to ensure they function as expected. See Testing policies.

About the Dashboard

The Dashboard displays a predefined statistical view of data that provides a snapshot of policy enforcement trends. Data is refreshed from the Activity Journal each time you open the Dashboard. This means that data is updated on demand. Also, if a section shows a statistic for the past week, that reflects not the last seven whole calendar days, but the last seven 24-hour periods starting from the top of the current hour.

Understanding Dashboard data

Table 3.2 summarizes the data presented in each of the predefined queries represented on the Dashboard, and discusses the possible implications of and uses of the data displayed in each.

Note: The system displays *No data available* if policies have not been deployed.

Table 3.2: Dashboard data

Information/ section	Description	Action required
Summary policy information at the top of the page	Policy status	• Review to identify anomalies.
Total Policies	The number of policies that have been added or edited in the past week (the last seven 24-hour periods).	• Review data to identify anomalies.
Deployed	The number of policies that have been deployed to policy enforcers in the past week. These policies are in the *Deployed* state.	• Review data to identify anomalies.
De-activated	The number of deployed policies that have been deactivated in the past week (the last seven 24-hour periods). On the *Policies* list, these policies have the state, *Inactive*. **Note**: The built-in policy tester feature cannot test policies that are in the *Inactive* state until those policies are deployed again.	• Review data to identify anomalies.
Draft Policies	Policies that are in the Draft state. Draft policies include: • *Inactive Draft*: Policies that have been added but have never been deployed. • *Deployed Draft*: Policies that have been deployed but are now in the draft state because they have been edited. The edited (draft) version has not yet been deployed.	• Deploy policies that are required and delete drafts that are no longer needed.
System configuration status	The current status of PDPs (Policy Decision Points). The information presented here is updated periodically: • **Active PDPs**: PDPs that have been active in the past week • **Heartbeat failed**: The number of heartbeat failures in the past week • PDPs up to date: The number of PDPs that are up to date • PDPs not up to date: The number of PDPs that are out of date	• Review the status and take action to resolve issues with PDPs.
System details	The status of the CloudAz subscription or trial, and information about the system configuration.	• Review the time remaining in the subscription or trial and renew them before they expire.
Activity alerts in last 7 days	The number of alerts that have been triggered by CloudAz policies in the past week. Alerts are configured in the Reports section.	• Identify policies that are being enforced at a high rate. Further review or action may be required.
PDP throughput in the last 24 hours	The number of requests sent to PDPs in the past 24 hours.	• Investigate high numbers of requests that might indicate performance issues.

Table 3.2: Dashboard data (Continued)

Information/ section	Description	Action required
Top 10 active users in last 7 days	The number of users that have triggered Allow or Deny actions in the past week.	• Identify incorrectly defined entitlements: users or group should have access, but policies are not updated or are incorrectly designed.
Top 10 activities by resource in last 7 days	A chart representing the five resources that any users have most frequently attempted to access and been allowed or denied by an active policy, in the past week.	• Identify resources of interest to users who should not be using them. • Identify incorrectly designed resource or user components that are blocking users who should have access.
Frequently evaluated policies in last 7 days	A chart representing the top ten policies evaluated most frequently.	• Educate users about access policies and individual or group entitlements, at a broad level. • Identify improperly designed policies that are blocking too many users who expect and are entitled to access or use.
Policies not matched in any request	A list of policies that have not been evaluated in response to any user actions.	• Review policies to ensure they are relevant and constructed properly.
Top 10 policies grouped by tags	A graphical representation of the tags related to the top ten policies that have been enforced in the past week.	• Review policies being watched by monitors that are tagged, and are being enforced at a high rate.
Recent activities	A list of actions that have been performed on CloudAz in the past week (the last seven 24-hour periods).	• Review activities to identify anomalies. • Identify improperly designed resource components or policies, that might allow inappropriate users access to sensitive resources.

Configuring CloudAz

To use NextLabs CloudAz to enforce policies, the system must be configured for your environment, and policies must be designed and deployed. Tasks include:

- **Identify policy requirements**: Identify the kind of resources and information you need to protect, the subjects or users who access those resources, and the policies you want to enforce in your environment. See Policy and Policy Model overview.
- **Configure permissions for CloudAz system users**: Create access rules or permission sets based on user attributes and conditions. See Adding and editing delegation policies.
- **Set up a Policy Model:** A Policy Model is one or more resource types that serve as templates for policy components. Resource types establish the attributes, actions, and obligations available to policy components, such as resource, action, and subject components. See Policy and Policy Model overview.
- **Configure policy objects, and attributes**: Policy objects, such as subject, action, and resource components, must be configured in the *Policies* section of the console. See Policy and Policy Model overview.
- **Write and deploy policies to PDPs**: After a Policy Model and policy objects are defined, the policies themselves can be written and deployed to PDPs (Policy Decision Points). See Policy and Policy Model overview.

  Note: Policies that have been deployed are enforced only after policy enforcers have been installed and configured.

- **Optional: Install and configure PEPs for additional applications**: Developers can create PEPs (Policy Enforcement Points) to enforce policies on applications that do not have predefined NextLabs enforcers.

Delegated administration

Delegated administration is a feature that enables administrators to control user access to the CloudAz user console by creating policies based on user attributes and conditions. This eliminates the need to create and apply roles to users or groups of users.

Delegated administration is very helpful when different types of users need to manage policies in the CloudAz console. For example, if two applications are secured using CloudAz policies, policy designers can restrict the administrators of one application from modifying the security policies of the other application by creating a delegation policy. See Adding and editing delegation policies.

Building PEPs and integrating them with CloudAz

Authorization decisions are enforced, and resources are secured, by PEPs (Policy Enforcement Points). To build a PEP and integrate it with CloudAz, developers must:

- Analyze and understand the authorization needs and environment
- Build a runtime integration by invoking the API
- Define the metadata required to create ABAC policies

Analyze and understand the authorization needs and environment

Before building a PEP or defining the metadata required to define policies, it's important to analyze and understand the authorization use cases that the system requires.

Procedure

1. Identify the objects that need to be secured and the circumstances under which they need to be secured. For example, if a content management system must be secured, determine which folders need to be secured, and identify the actions that need to be controlled, such as copy, download, and email.
2. List all the attributes that the object provides. These attributes can be used in the ABAC policy.
3. Identify the interception point in the application where you can introduce the authorization check. For example, if the authorization is for data in a web application, the interception to perform the authorization check must occur before the data is queried. After the authorization check is performed, the application performs a query rewrite and runs the new query. See Integration patterns with CloudAz.

4 Identify various sources of attributes, or attribute providers, to be used in ABAC policies.

5 Understand where the Subject attributes, which are the attributes related to the user accessing the application, are retrieved. For example if the application is protected by SAML (Security Assertion Markup Language) authentication, user attributes are part of SAML assertions. Other examples of attribute resources include HR systems and custom user databases.

6 Identify how the application uniquely represents the user. In other words, identify the attributes that identify a given user.

7 Identify any other user or environment attributes to be used in the policy. These attributes might include the time of the day or the authentication mechanism of the user.

8 Prepare the development environment to build a PEP in the language of your choice.

Build a runtime integration by invoking the API

Developers can build a runtime integration by invoking the API.

Before you begin

- Conduct a needs analysis to identify the authorization requirements and environment. See Analyze and understand the authorization needs and environment.
- Sign up for the CloudAz service, activate your account, and use the QuickStart guide to learn the basics of the service. To access the QuickStart guide, click **Getting Started** under the Help icon at the upper right of the CloudAz console, then click the QuickStart link.

Procedure

1 Download the CloudAz client SDK for the programming language you want to use from https://s3-us-west-2.amazonaws.com/nxtlbsrelease/Platform_SAAS/openAZ-pep/Next-labs-OpenAZ-PEP.zip.

Note: If there is no client SDK for the language of your choice, use the REST APIs directly. See REST API Reference.

2 Implement the code that invokes the authorization API: Pass all the attributes required for the policy evaluation, including the subject, resource, and environment attributes. Use the attribute short names when invoking the authorization API.

3 Process the response to enforce the authorization decision in the application.

4 Process obligations. For example, if the PDP returns an obligation to notify the user, add code to your application to display the notification.

Define the metadata required to create ABAC policies

Use the CloudAz console to define the metadata required to create ABAC policies.

Before you begin

- Conduct a needs analysis to identify the authorization requirements and environment. See Analyze and understand the authorization needs and environment.
- Sign up for the CloudAz service, activate your account, and use the QuickStart guide to learn the basics of the service. To access the QuickStart guide, click **Getting Started** under the Help icon at the upper right of the CloudAz console, then click the QuickStart link.

Procedure

1 Log in to CloudAz.

2 Add the subject attributes identified in the needs analysis:

a On the left navigation bar, click **Policy Model**.

b On the *Policy Models* page, double-click the User Policy Model.

c In the *Attributes* section, add the user attributes to be used in policies.

Note: Developers use the attribute short names when invoking the authorization API.

3 Add the subject components to be used in policies:

d On the left navigation bar, click **Components > Subject**.

e Click **Add Component**, then provide the subject properties. See Adding and editing policy components.

Note: Developers use the attribute short names when invoking the authorization API.

4 Create a resource type that represents the object being secured:

a On the left navigation bar, click **Policy Model**.

b Click **Add Resource Type**, then provide the Resource Type properties. See Adding and editing resource types.

Note: Developers use the attribute short names when invoking the authorization API.

5 Create ABAC policies using the Policy Model and components. See Adding policies.

6 Deploy the components and policies. See:

- Deploying components
- Deploying policies

7 Run the application to test your enforcement results.

ABAC Use Cases & Integration

Use cases & integration patterns

This section describes the key concepts of ABAC (Attribute Based Access Control) and provides guidelines for ABAC implementation. These guidelines have been consistently applied in many organizations with considerable success. They do not, however, constitute a comprehensive guide for all authorization scenarios.

This section also provides high-level steps developers should consider taking before building an ABAC integration to enforce authorization decisions.

Topics in this section

Business need for authorization

There are multiple scenarios where authorization is required in an application. For example:

- **Secure access to data in any application**: Application data can be structured or unstructured, and authorization is often required for data access. For example, a patient care system has patient health records that are protected by various laws and regulations. This system can be accessed from traditional browser-based interfaces or mobile interfaces. In both cases, data access requires the same authorization.
- **Secure business transactions**: In enterprise software systems such as CRM (Customer Relationship Management), ERP (Enterprise Resource Planning), or HCM (Human Capital Management), there are different types of business transactions. Some of these transactions are sensitive and business rules define who can perform the transactions.
- **Secure services, APIs, and endpoints**: In complex enterprises, distributed systems must interact to achieve the desired results. For example, an e-commerce service application must interact with other services such as order management systems, customer management systems, inventory management systems, and payment management systems. Typically, interactions within distributed systems are performed as a system user or an elevated privileges user. In some scenarios, legacy systems might not have built-in security features. In these cases, regardless of the backend services, the final e-commerce service or API still requires authorization. That service or API must determine whether users can invoke the service, and whether data can be exposed users.
- **Secure information sharing**: In this connected world, information sharing is paramount for many day-to-day transactions. Much of the information being shared is unstructured and contained in email, documents, and secure messages. This information is primarily secured during transportation using secure network protocols and encryption. However, additional measures are required to ensure that only authorized users receive the appropriate email, documents, and messages.

- **Other scenarios**: In government, military, and high-tech development sectors, data sensitivity depends on multiple parameters. These parameters include the location from which data is being accessed, the clearance level of users accessing the data, and national security advisory levels. In these scenarios, security policies must change dynamically and authorization decisions need to be based on real-time information.

Use cases addressed by ABAC

Within each of above scenarios there are various use cases where fine-grained or coarse-grained authorization is required. These use cases are described in Table 4.1.

Table 4.1: Use cases addressed by ABAC

Business scenario	Authorization use cases
Secure access to data in any application	• Restrict the rows of data users can access in an application • Redact sensitive columns of data for unauthorized users • Restrict the operations users can perform on specific data • Protect structured and unstructured data
Secure business transactions	• Control the operations users can perform in applications • Control the UI widgets that are visible in web applications • Control the display of menus and menu items in applications • Display or hide buttons that submit forms in user interfaces
Secure services, APIs, and endpoints	• Protect API gateways, micro services, web services, REST APIs, and SOA infrastructure • Control the services users can invoke • Protect legacy services
Secure information sharing	• Restrict the recipients of email based on the content or attachment • Control access to sensitive discussion forums
Other scenarios	• Perform authorization using geospatial data

Integration patterns with CloudAz

This section describes patterns of integration used with CloudAz.These patterns include:

- Data security: Data filtering for relational data
- Data redaction
- Authorization for content management and unstructured data
- Secure business transactions
- Portals and web applications
- Micro services, APIs, and endpoints
- Authorization using geospatial data

Data security: Data filtering for relational data

Consider the example of an HR (human resources) application that provides various features such as managing employee benefits, searching for other co-workers in the organization, and so on. For this application, the employee directory is stored as a relational database. The typical authorization use case is:

> Allow managers to edit only information of employees reporting in their hierarchy

There are two methods of handling this authorization. The first is computationally intensive; the second is a better way to handle the authorization:

- Query the database for all the records and then apply authorization checks for every single row before displaying the subset of data. This approach requires a significant amount of pre- and post-computation to display the results.
- Use QueryRewrite with security filters and obligations in an ABAC policy that has these properties:
  - **Policy Effect**: Allow
  - **Subject**: Users who have the role attribute of Manager
  - **Resource**: Employee object
  - **Action**: Edit Employee object
  - **Obligation Name**: QueryRewrite
  - **Obligation Value**: WHERE MANAGER = :CURRENT_USER

  When the HR application makes an authorization request:

  Note: This is example code only. For the required syntax, see REST API Reference.

```
PepAgent pepAgent = pepAgentFactory.getPepAgent();
user.addattribute("role","Manager");
user.addattribute("username","jdoe");
Action action = new Action("EDIT EMPLOYEE");
Resource resource = new Resource("Employee");
pepResponse = pepAgent.decide(user, action, resource, application);
String whereClause = pepResponse.getObligations("QueryRewrite");

// use the whereClause in appending to the SQL clause before executing it
String sqlQuery = "Select emp_name, dob, salary, job_title from emp_table";
sqlQuery.append (whereClause);

//execute the query
stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

The results that are returned to the UI are prefiltered based on authorization checks. This also eliminates the tight coupling between security policies and business logic. If there is no security guidance, administrators can modify the policy and achieve the result shown in Figure 4-1.

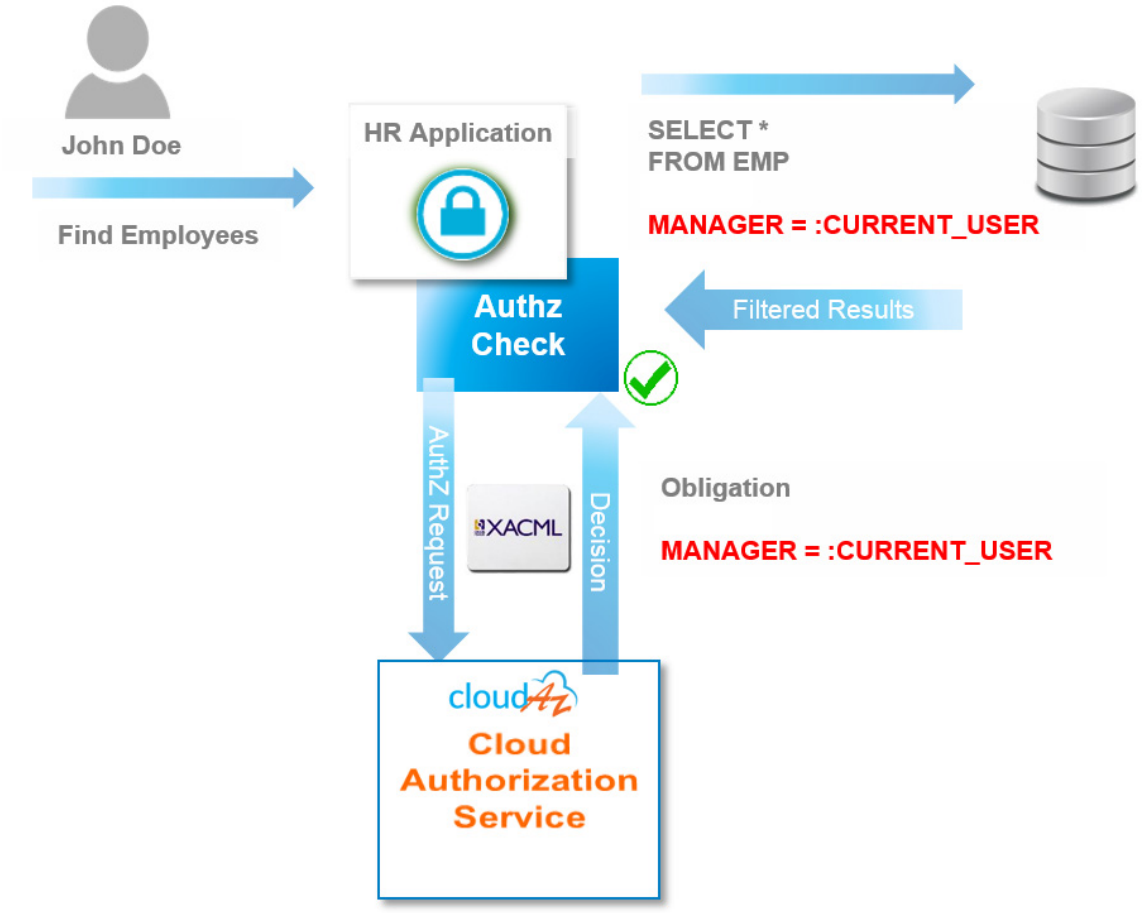


Figure 4-1: Integration patterns

Data redaction

Another common use case in data security is data redaction: masking or suppressing sensitive data such as personally identifiable information. For example, managers should be able to view employee records, but the employee date-of-birth column should be redacted.

To accomplish this, one or more obligations can be added to the policy shown in Data security: Data filtering for relational data:

- **Policy Effect**: Allow
- **Subject**: Users whose role attribute is Manager
- **Resource**: Employee object
- **Action**: Edit Employee object
- **Obligation Name**: Query Rewrite
- **Obligation Value**: MANAGER = :CURRENT_USER
- **Obligation Name**: Redact Data

- **Obligation Value**: Date of Birth

In the application, the developer handles the second obligation by modifying the SQL query by using the decode function and passing the masking characters.

Authorization for content management and unstructured data

In general, content management systems provide a great deal of flexibility for storing, categorizing, and managing documents and content. This flexibility, however, often leaves content vulnerable to unauthorized access or requires sophisticated security schemes for access control.

The content itself is contained in a variety of formats such as documents, spreadsheets, and CAD drawings.

Typical authorization use cases for content management systems include:

- Restrict access to folders that are created for a specific project
- Hide files in a folder if the classification of the file is greater than the classification of the user logged in

The pattern to solve these kind of use cases requires a deeper integration with the content management system. Developers must enforce ABAC policies by using the content management system's APIs to filter the files or hide the folders.

The following is a sample ABAC policy for content management systems:

```
Allow Users
TO view and edit documents
IF   (user.secClearance      => doc.sensitivity
  AND user.citizenship       =  doc.reqdCitizenship
  AND user.projectMembership =  doc.project)
OR   (doc.sensitivity        =  PUBLIC)
```

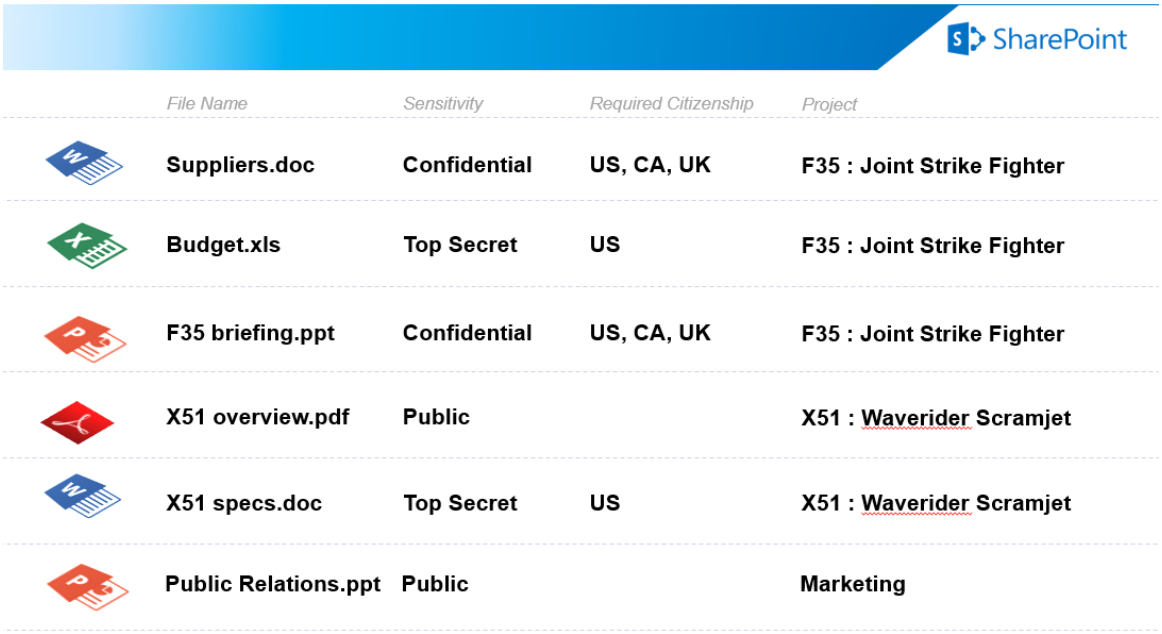


Figure 4-2: A Sample rendering of a SharePoint content management system

Secure business transactions

Consider an example of a supply chain management application. Multiple personas interact with the application and there are many different types of business transactions that require authorization. Consider the following example:

- Allow full-time employees to *Create* purchase orders for office supplies

In this example *Create* is a business transaction that can be represented by a button in the user interface. The authorization check determines when to hide or show that button.

A sample ABAC policy would be as follows:

- **Policy Effect**: Allow
- **Subject**: Users who are full-time employees (Employment Status = FT)
- **Resource**: Purchase Order (PO.Type = Office supplies)
- **Action**: Create

A simple AngularJS implementation for the policy would be as follows:

```
$scope.decision = NextLabsPEPAgent.decide (mySubject, resource, action,
application);

//Bind the hide or show (ng-show) property of the button to the result
<form ng-show= "decision" ng-submit="addTodo()">
    <input class="btn-primary" type="submit" value="add">
</form>
```

The **Create** button would be shown only if the policy evaluation returned true.

Portals and web applications

A portal typically combines widgets and content from multiple sources. Developers can define fine-grained access control rules using ABAC to restrict access to different widgets. Similarly, many enterprise applications have navigation menus, submenus, regions, tables, and various other web components that should be displayed only to authorized users as shown in Figure 4-3.

In ABAC policies, developers can model all these artifacts as resources and define very fine-grained authorization rules and enforce the authorization.

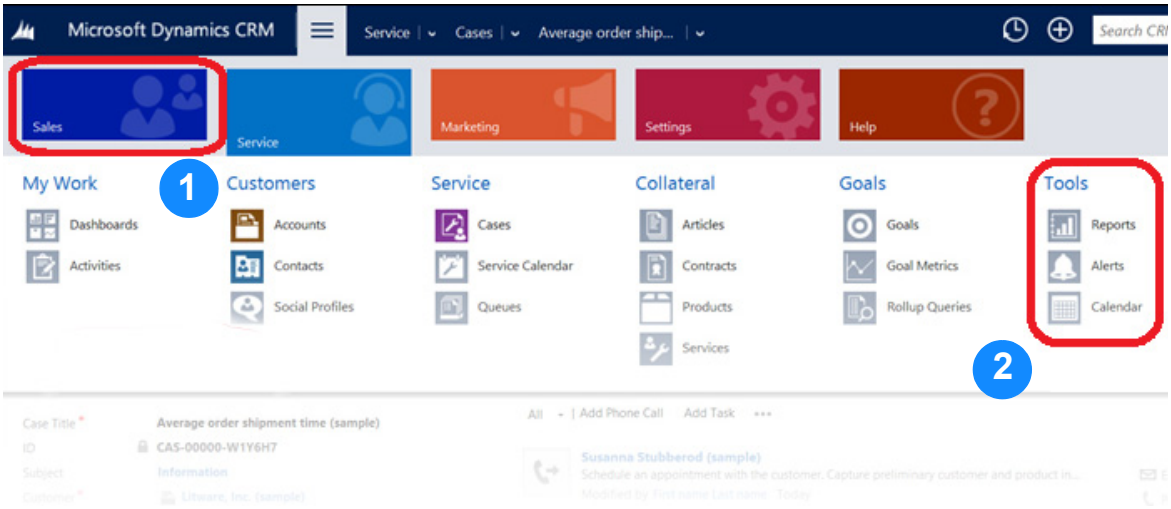


Figure 4-3: Integration patterns for portals and web applications

- 1: **Sales links**: Displayed to Sales team only
- 2: **Tools links**: Displayed to appropriate users based on attributes

Micro services, APIs, and endpoints

API gateways are powerful and modern tools to extend business functionality. They help to combine multiple backend services (internal and external) and build a simple façade. In this hybrid system, authorization is the responsibility of the façade that is exposing all the backend services.

Other patterns with API gateways virtualize existing legacy systems to work with newer systems, such as mainframe legacy application services that are required for mobile applications. For example, a supplier management system might have two different services providing the data: a supplier service and an order management service.

Typical authorization requirements in this scenario include:

- **API access: Only authorized users can invoke the API**: Users in the Finance and Marketing departments should be able to query supplier data and access the Supplier API.

- **Data filtering and redaction: What data will the API expose to the user**: Users in the Marketing department can view suppliers in their own region only, but those users are not authorized to view bank account numbers.

A sample ABAC policy would be as follows:

- **Policy Effect**: Allow
- **Subject**: Finance department users (Department = Finance)
- **Resource**: Suppliers in a given region; Data (`Supplier.location = :logged_in_user.loc`)
- **Action**: Query

The enforcement point for this request occurs before the response leaves the API gateway. The developer checks with the PDP to determine whether a particular row is authorized to be in the response. If it is not authorized, the row is removed. Figure 4-4 shows an overview of the integration. In the sample architecture, CloudAz is working with a cloud-based API gateway.

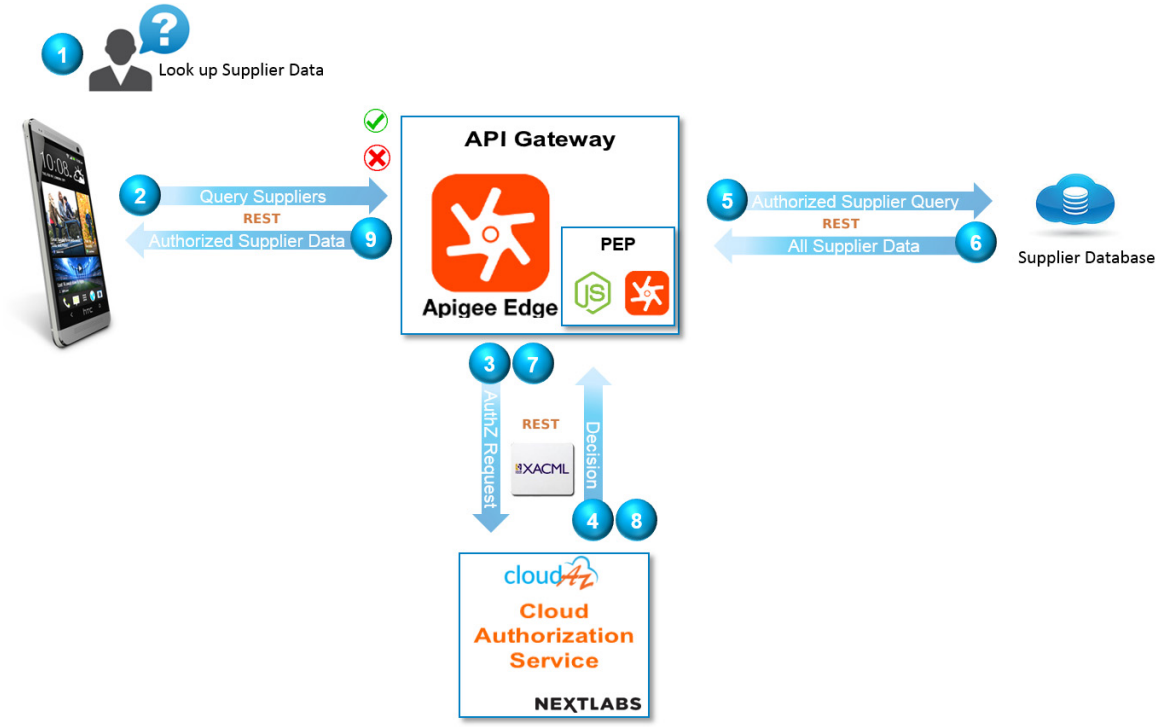


Figure 4-4: Integration patterns for micro services APIs and endpoints

A simple JavaScript implementation of this would be as follows:

Note: This is example code only. For the required syntax, see Using the JavaScript client library.

```
//Build OpenAz request by gathering all user and resource attributes including action
//supplier is a representation of a row supplier information
//supplierinfo is array of all the object retrieved from backend

var supplierInfo = context.proxyResponse.content;
  for (var supplierName in supplierInfo) {
    if(supplierInfo.hasOwnProperty(supplierName)) {
      var supplier = supplierInfo[supplierName];
     var decision = NextLabsPepAgent.decide(mySubject,supplier,action,application);
 if (decision != true){

//remove the supplier information into response payload
delete supplierInfo[supplierName];
}
}
```

Authorization using geospatial data

In organizations that deal with logistics, many access control decisions are based on the locations from which data is being accessed or the location of the resource itself. For example, in the shipping industry, a cargo operator based in a port in Singapore should be able access all the cargo information of ships entering Singapore territorial waters. For other ships in the area, however, the operator should be able to access only the ship's name and location. All other data should be redacted, as shown in Figure 4-5.

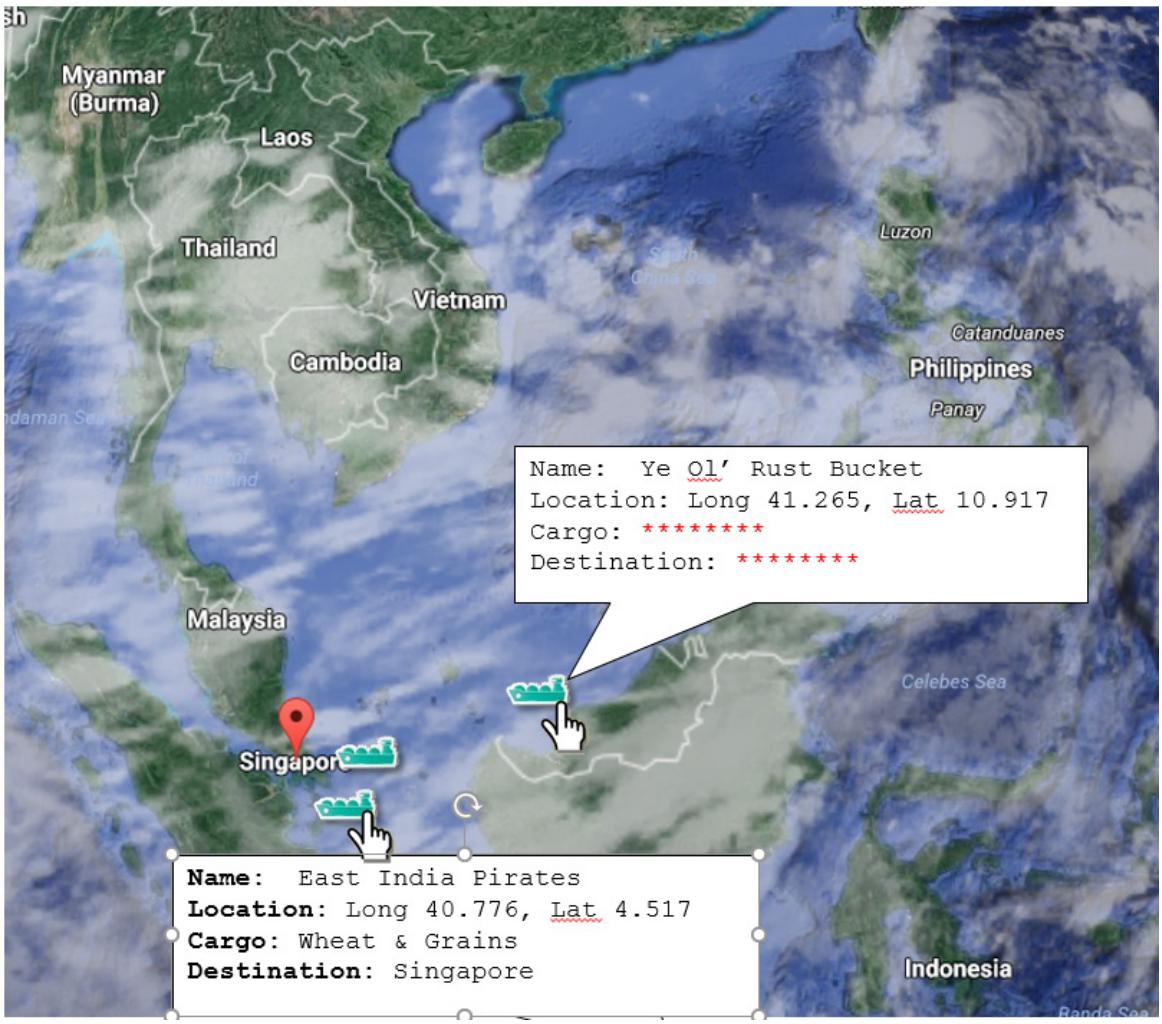


Figure 4-5: Geospatial data

In this scenario developers can use CloudAz ABAC policies to define geospatial data, such as the longitude and latitude of the protected area, as user attributes. These attributes can then be compared with the geolocation of ships to determine access control rights of the user.

There are many similar use cases where geographical information obtained from mobile devices could be used to determine the access control rights of users. For example, organizations with field service operations could restrict access to customer data based on the current location of their employees.

SDK & OpenAz Client Libraries

SDK and OpenAz Client Libraries

This section explains how to make authorization requests based on OpenAz using Java and JavaScript client libraries. It also provides a quick reference for the SDK.

Topics in this section

About the OpenAz PEP client SDK

OpenAz is an open source initiative that provides a standards-based interface. This interface enables developers to interact with an Attribute Based Access Control (ABAC) system. The OpenAz PEP client SDKs provide an abstraction layer that PEP (Policy Enforcement Point) developers can use to enforce authorization. In addition, if you are using the Java client SDK, this abstraction layer enables you to switch seamlessly among the PDPs (Policy Decision Points) of various ABAC providers. NextLabs provides Java and JavaScript client SDKs.

Using the Java client library

This section explains how to use NextLabs OpenAz PEP Java client to construct and send authorization requests to remote PDPs, including REST PDPs and Cloud PDPs.

Setting up the Java SDK

If you are developing Java PEPs, add the required JAR libraries to your Java project to set up the Java SDK.

Procedure

1. Set up the Java SDK with JDK 7 or higher. For instructions, go to http://docs.oracle.com/javase/7/docs/webnotes/install/.
2. Download the sample Java application: https://s3-us-west-2.amazonaws.com/nxtlbsrelease/Platform_SAAS/openAZ-pep/Nextlabs-OpenAZ-PEP.zip.

3 Extract the files from the package.

Note: Files are extracted with the *Read-only* attribute set in their properties. To view or change file properties in Windows, right-click the file, then select *Properties*.

4 Add `nextlabs-openaz-pep.jar` and the required JAR libraries to your Java project. These libraries are located in the `lib` folder.

Next steps

Invoking the PDP

Invoking the PDP

To invoke the PDP, you must configure the Java client SDK as described in this section.

Configuring the Java client SDK to invoke remote or CloudAz PDPs

For remote or CloudAz PDPs, authorization requests are sent through a `PEPAgent`, which is created and configured using a `PEPAgentFactory`. The `PEPAgentFactory` is configured with properties that can come from either a file or a Properties object. To use a file, create the file with the appropriate configuration for your system, then reference that file in the Java code. To use a Properties object, include the properties in the Java code directly.

Note: Embedded PDPs are not supported in this version of CloudAz.

Procedure

1 Specify the `PDPEngine` name to initialize the PDP. For example:

```
# PDP Engine configurations
#       - Embedded PDP:       com.nextlabs.openaz.pdp.EmbeddedPDPEngine
#       - REST/ CloudAz PDP: com.nextlabs.openaz.pdp.RestPDPEngine
#------------------------------------------------------------------------------
# PDPEngine class

# When using REST/ CloudAz PDP, configure the properties below


# The host name of the CloudAz server, for example: saas-jpc.cloudaz.com
#------------------------------------------------------------------------------
nextlabs.cloudaz.host=<CloudAz REST API host>

# The port on which the CloudAz service is listening on the server
#------------------------------------------------------------------------------
nextlabs.cloudaz.port=443

# Whether the CloudAz service is over https (true or false)
#------------------------------------------------------------------------------
nextlabs.cloudaz.https=false

# The authentication settings to connect with the REST/CloudAz service
# Two authentication types are available:
#       - No authentication required: NONE
#       - Use with OAuth2 provided authentication service: OAuth2
```

```
#-------------------------------------------------------------------------------
nextlabs.cloudaz.auth_type=OAUTH2

#
# OAuth2 Related configurations
#  Use only if nextlabs.cloudaz.auth_type is OAUTH2

# The OAuth2 Authorization Grant Type
#  The available grant type is
#       - client_credentials (default)
#-------------------------------------------------------------------------------
nextlabs.cloudaz.oauth2.grant_type=client_credentials

# The OAuth2 server configurations
#  If the default OAuth2 service is provided by Control Center server,
#  configure the Control Center server host and port
#-------------------------------------------------------------------------------
nextlabs.cloudaz.oauth2.server=<control center host>
nextlabs.cloudaz.oauth2.port=<control center port>
nextlabs.cloudaz.oauth2.https=true

# The client ID required to identify the client connecting over OAuth2
#-------------------------------------------------------------------------------
nextlabs.cloudaz.oauth2.client_id=<CLIENT_ID>

# The client secret for the OAuth2 client credentials grant
#-------------------------------------------------------------------------------
nextlabs.cloudaz.oauth2.client_secret=<CLIENT_SECRET>

# The OAuth endpoint to get the token for the client credential grant
#   - CloudAz endpoint: /oauth/token
#   - REST endpoint   : /cas/token
#-------------------------------------------------------------------------------
nextlabs.cloudaz.oauth2.token_endpoint_path=/cas/token

# Ignore HTTPS self-signed certificate errors if using self-signed certificates
#-------------------------------------------------------------------------------
nextlabs.cloudaz.ignore_https_certificate=true
```

2 To use a Properties object, include the properties in the Java PEP client code directly as follows:

Note: As a good practice, use a Credential Store Framework to securely store and retrieve the `client_secret` key.

```
Properties xacmlProperties = new Properties();
// The name of the CloudAz/ REST PDP server host
xacmlProperties.setProperty(Constants.PDP_REST_HOST, "<CloudAz REST API host>");

// The OAuth2 server configurations
xacmlProperties.setProperty(Constants.PDP_REST_PORT, "443");
xacmlProperties.setProperty(Constants.PDP_REST_HTTPS, "true");
xacmlProperties.setProperty(Constants.PDP_REST_AUTH_TYPE, "OAUTH2");
xacmlProperties.setProperty(Constans.PDP_REST_OAUTH2_TOKEN_GRANT_TYPE,
"client_credentials");

// The client ID required to identify the client connecting using OAuth2
xacmlProperties.setProperty(Constants.PDP_REST_OAUTH2_CLIENT_ID, "<CLIENT_ID>");

// The client secret for the OAuth2 client credentials grant
```

```
xacmlProperties.setProperty(Constants.PDP_REST_OAUTH2_CLIENT_SECRET,
"<CLIENT_SECRET>");

// The OpenAz API PDP engine factory implementation
xacmlProperties.setProperty(XACMLProperties.PROP_PDPENGINEFACTORY,
"com.nextlabs.openaz.pdp.PDPEngineFactoryImpl");

// The OpenAz API configuration
xacmlProperties.setProperty(Constants.ENGINE_NAME,
"com.nextlabs.openaz.pdp.RestPDPEngine");

// Mapper classes used internally to map requests
xacmlProperties.setProperty("pep.mapper.classes",
"com.nextlabs.openaz.pepapi.RecipientMapper,com.nextlabs.openaz.pepapi.Discretionary
PoliciesMapper,com.nextlabs.openaz.pepapi.HostMapper,com.nextlabs.openaz.pepapi.Appl
icationMapper");
```

3 To use a properties file, create the file with the appropriate configuration for your system, then reference that file in the Java code as follows:

```
import org.apache.openaz.pepapi.std.StdPepAgentFactory;
import org.apache.openaz.pepapi.PepAgent;
PepAgentFactory pepAgentFactory = new StdPepAgentFactory("/path/to/properties.file");
PepAgent pepAgent = pepAgentFactory.getPepAgent();
```

Making authorization requests using Java

PEPs (policy enforcement points) send authorization requests to PDPs (policy decision points) and receive authorization decisions in response. To make authorization requests using Java PEPs, use the NextLabs PEPAgent.

Procedure

1 Initialize the PEPAgent and establish the PEPAgent secured connection with the PDP:

```
// Using a properties file
PepAgentFactory pepAgentFactory = new StdPepAgentFactory("/path/to/
properties.file");
PepAgent pepAgent = pepAgentFactory.getPepAgent();

// Using a properties object
PepAgentFactory pepAgentFactory = new StdPepAgentFactory(xacmlProperties);
PepAgent pepAgent = pepAgentFactory.getPepAgent();
```

2 Build the request:

a **(Required)** Build the Subject object with a unique ID. Typically this is a SID (Windows security identifier or UNIX ID) or an email address. Additional attributes can be populated using `addAttribute`.

```
    Subject user = Subject.newInstance("chris.webber@hdesk.com");
    user.addAttribute("user_id", "chris.webber");
```

b **(Required)** Build the Action object using the Action short name. The short name must match that of the Action resource type.

```
    Action action = Action.newInstance("VIEW_TKTS");
```

c **(Required)** Build a Resource object with a resource ID, which is typically the name of the resource, and populate all the available resource attributes.
`ID_RESOURCE_RESOURCE_TYPE` is a required attribute. The value of this attribute must match the short name of the resource type:

```
    Resource resource = Resource.newInstance("Ticket:1103");
    resource.addAttribute(Constants.ID_RESOURCE_RESOURCE_TYPE.stringValue(),
"support_tickets");
```

d **(Required)** Build an Application object:

```
    Application application = Application.newInstance("Outlook");
```

e **(Optional)** Build an Environment object with environment attributes:

```
    Environment environment = Environment.newInstance();
    environment.addAttribute("authentication_type", "multifactor");
```

f **(Optional)** If you are enforcing policies on an application that sends email, such as Microsoft Outlook, build the Recipient object.The Recipient object can include a single email address or a list of email addresses. However, attributes can be associated with a single recipient only. If your policies depend on attributes, and you need to specify multiple recipients, each recipient must be placed in a separate request:

```
    Recipient recipient = Recipient.newInstance ("frank.underwood@whitehouse.gov",
"claire.underwood@whitehouse.gov", "doug.stamper@whitehouse.gov");

    Recipient recipient = Recipient.newInstance ("frank.underwood@whitehouse.gov");
recipient.addAttribute("title", "President");
```

3 Send the authorization request from the PEPAgent using the `decide()` method:

Note: The order of arguments (user, action, resource, environment, and recipient) does not matter. Arguments can be specified in any order. Also, only subject, action, and resource are required.

```
   // Submit the authorization request
   PepResponse pepResponse = pepAgent.decide(subject, action, resource, application,
environment, recipient);
```

Using the JavaScript client library

This section explains how to use NextLabs OpenAz PEP JavaScript client to construct and send an authorization request to the NextLabs REST API and how to process its response.

Setting up the JavaScript SDK

To use the JavaScript client library, you must set up the JavaScript SDK and configure NextLabs OpenAz as a dependency in `package.json`.

Procedure

1 Set up the JavaScript SDK. For instructions, go to https://nodejs.org/en/download/.

2 Verify that the JavaScript node version is 4.5 or higher, and the npm version is 3.10.6 or higher. To check version numbers, use these commands:

```
node --version
npm --version
```

3 Download the sample JavaScript application from https://s3-us-west-2.amazonaws.com/nxtlbsrelease/Platform_SAAS/openAZ-pep/Nextlabs-OpenAZ-PEP.zip.

4 Extract the files from the package.

5 Open the extracted folder and go to the `js` folder, which contains the required files and samples for the NodeJS API.

Note: Files are extracted with the *Read-only* attribute set in their properties. To view or change file properties in Windows, right-click the file, then select *Properties*.

6 Clone the project, then specify NextLabs OpenAz as a dependency in `package.json` using a local path:

```
"dependencies": {
    "nextlabs-openaz":"path/to/project",
    ...other dependencies...
}
```

7 Install the module using the following command:

```
npm install
```

8 To use the module:

```
var nextlabsopenaz = require("nextlabs-openaz");
```

Next steps

Load required properties. See Required properties and Making authorization requests using JavaScript.

Required properties

Properties can be loaded through `openAz-pep.json` or as a JavaScript object. The following properties are required to connect to the REST API:

- `nextlabs.cloudaz.host`: The host name of the server where the NextLabs Policy Decision Point is located. For example, `jpc.localdomain`.
- `nextlabs.cloudaz.port`: The port used by Tomcat or JBoss for REST API connections. For example, `58080`.
- `nextlabs.cloudaz.https`: The protocol used for REST API connections. For HTTPS use `false`.
- `nextlabs.cloudaz.auth_type`: The authorization method for connecting to the REST API. For example, `OAUTH2`.
- `nextlabs.cloudaz.oauth2.grant_type`: The authentication grant type for REST API connections. For example, `client_credentials`.

- `nextlabs.cloudaz.oauth2.client_id`: The `client_id` credential required for authentication with the application. This is the username of the account created for the connection. See Configuring user authentication for the REST API.
- `nextlabs.cloudaz.oauth2.client_secret`: The `client_secret` credential required for authentication with the application. This is the password of the account created for the connection. See Configuring user authentication for the REST API.
- `nextlabs.cloudaz.ignore_https_certificate`: Whether to ignore the HTTPS certificate warnings (`true`) or not (`false`). If you are using a self-signed SSL certificate, use the value `true`.
- `nextlabs.cloudaz.oauth2.server`: The hostname of the server where the Control Center is installed. For example, `<Control_Center_host>`.
- `nextlabs.cloudaz.oauth2.port`: The port to use for REST API connections. For HTTPS, use `443`.
- `nextlabs.cloudaz.oauth2.https`: The HTTPS connection for the REST API. For HTTPS, use `true`.
- `nextlabs.cloudaz.oauth2.token_endpoint_path`: The Control Center host name or IP address and the authentication token. For example, `<Control_Center_host>/cas/token`.

Example

```
{
"nextlabs.cloudaz.host": "<Java PC host> e.g.: jpc.localdomain",
"nextlabs.cloudaz.port": "58080",
"nextlabs.cloudaz.https": false,
"nextlabs.cloudaz.auth_type": "OAUTH2",
"nextlabs.cloudaz.oauth2.grant_type": "client_credentials",
"nextlabs.cloudaz.oauth2.client_id": "<CLIENT_ID>",
"nextlabs.cloudaz.oauth2.client_secret": "<CLIENT_SECRET>",
"nextlabs.cloudaz.ignore_https_certificate": true,
"nextlabs.cloudaz.oauth2.server": "<Control_Center_Host>",
"nextlabs.cloudaz.oauth2.port": "443",
"nextlabs.cloudaz.oauth2.https": true,
"nextlabs.cloudaz.oauth2.token_endpoint_path": "/cas/token"
};
```

Making authorization requests using JavaScript

PEPs (policy enforcement points) send authorization requests to PDPs (policy decision points) and receive authorization decisions in response. To make an authorization request using JavaScript, you need to load properties, build the request, and send the request to the PDP from the NextLabs PEPAgent, an instance of which is created with the `new` operator. For more information, see Required properties.

Procedure

1 Load the required properties using a JavaScript object or a JSON file:

To load properties using a JavaScript object:

```
var pepAgent = new NextLabsPEPAgent(openazProperties);
```

To load properties using a JSON file and establish a secure connection:

```
var openazProperties = require ("./config/openaz-pep.json");
```

2 Initialize the PEPAgent and establish the PEPAgent secured connection with the PDP:

```
var nextlabsopenaz = require("nextlabs-openaz");
var NextLabsPEPAgent = nextlabsopenaz.NextLabsPEPAgent;

var openazProperties = require("./config/openaz-pep.json");
var pepAgent =  new NextLabsPEPAgent(openazProperties);
```

3 Build the request:

a **(Required)** Build the Subject object with a unique ID. Typically this is a SID (Windows security identifier or UNIX ID) or an email address. Additional attributes can be populated using `addAttribute`:

```
var Subject = nextlabsopenaz.Subject;
var user = new Subject("chris.webber@hdesk.com");
user.addAttribute("user_id", "chris.webber");
```

b **(Required)** Build the Action object using the Action short name. The short name must match that of the Action resource type:

```
var Action = nextlabsopenaz.Action;
var action = new Action("VIEW_TKTS");
```

c **(Required)** Build a Resource object with a resource ID, which is typically the name of the resource, and populate all the available resource attributes.
`ID_RESOURCE_RESOURCE_TYPE` is a required attribute. The value of this attribute must match the short name of the resource type:

```
var Resource = nextlabsopenaz.Resource;
var resource = new Resource("Ticket:1103");
resource.addAttribute(NextLabsXACML.ID_RESOURCE_RESOURCE_TYPE,
"support_tickets")
resource.addAttribute("ticket_id", "1103");
```

d **(Required)** Build the Application object and its attributes:

```
var Application = nextlabsopenaz.Application;
var application = new Application("HelpDeskApp");
application.addAttribute("version", "1.0");
```

e **(Optional)** Build an Environment object and its attributes:

```
var Environment = nextlabsopenaz.Environment;
var environment = new Environment();
environment.addAttribute("authentication_type", "multifactor");
```

f **(Optional)** Specify the host or the machine that processes authorization requests. To create an instance of Host, pass an integer as an IP address or a string as a hostname, whichever is most convenient. No additional attributes are supported:

```
// default localhost
var host = new Host();

// using host name
var host1 = new Host("some_host");

// using inet address
var host2 = new Host(0x7f000001);
```

g **(Optional)** If you are enforcing policies on an application that sends email, such as Microsoft Outlook, build the Recipient object.The Recipient object can include a single email address or a list of email addresses. However, attributes can be associated with a single recipient only. If your policies depend on attributes, and you need to specify multiple recipients, each recipient must be placed in a separate request:

```
    // single recipient with id and email
    var recipient = new Recipient("abc@gmail.com", "abc");

    // multiple recipients with email addresses
    var recipient2 = new Recipient("abc@gmail.com", "def@gmail.com");
```

4 Send the authorization request from the PEPAgent using the `decide()` method:

```
   // Include the details required for the decide method: subject, action,
   // resource, application
   // Include optional details as needed: environment, host, recipient
   // pepAgent.decide(String subject, String action, String resource,
   // String environment, String application, String recipient)

   pepAgent.decide(subject, action, resource, application, environment, host,
recipient,).then(function(pepResponse) {
      assert(pepResponse instanceof PEPResponse);

  });
```

Configuring user authentication for the REST API

The CloudAz REST API uses OAuth 2.0 for authentication. To help you get started quickly, the CloudAz instance comes preconfigured with `client_id` and `client_secret` credentials. You can add additional credentials to your instance and change authentication settings as described in this section.

Procedure

1 Set authentication properties:

- For the Java PEP SDK, specify the following properties in the `openaz-pep.properties` file:

```
nextlabs.cloudaz.host=<JPC's host>

# JPC's REST Port
nextlabs.cloudaz.port=58080

# JPC is not using HTTPS
nextlabs.cloudaz.https=false

# Auth type is same as CloudAz
nextlabs.cloudaz.auth_type=OAUTH2

# Need to specify client_id and client_secret, as for CloudAz
nextlabs.cloudaz.oauth2.grant_type=client_credentials
nextlabs.cloudaz.oauth2.client_id=<CLIENT_ID>
nextlabs.cloudaz.oauth2.client_secret=<CLIENT_SECRET>
```

```
# Ignore HTTPS self-signed certificates error
nextlabs.cloudaz.ignore_https_certificate=true

# The OAuth2 server is Control Center
nextlabs.cloudaz.oauth2.server=<Control Center Host>
nextlabs.cloudaz.oauth2.port=443
nextlabs.cloudaz.oauth2.https=true

# The endpoint must be specified explicitly and it's different from CloudAz's
# endpoint which is /oauth/token
nextlabs.cloudaz.oauth2.token_endpoint_path=/cas/token
```

- For the JavaScript PEP SDK, specify the following properties in the `openaz-pep.json` file:

```
{
  "nextlabs.cloudaz.host": "<Java PC host> eg: jpc.localdomain",
  "nextlabs.cloudaz.port": "58080",
  "nextlabs.cloudaz.https": false,
  "nextlabs.cloudaz.auth_type": "OAUTH2",
  "nextlabs.cloudaz.oauth2.grant_type": "client_credentials",
  "nextlabs.cloudaz.oauth2.client_id": "<CLIENT_ID>",
  "nextlabs.cloudaz.oauth2.client_secret": "<CLIENT_SECRET>",
  "nextlabs.cloudaz.ignore_https_certificate": true,
  "nextlabs.cloudaz.oauth2.server": "<Control Center Host>",
  "nextlabs.cloudaz.oauth2.port": "443",
  "nextlabs.cloudaz.oauth2.https": true,
  "nextlabs.cloudaz.oauth2.token_endpoint_path": "/cas/token"
}
```

2 In the CloudAz console, create a user account for the PEP client with the following settings:

- Username = The `client_id` specified in the properties file.
- Password = The `client_secret` specified in the properties file.
- Add the user attribute `jwt_passphrase`. The value of this attribute can be anything except for the password (client_secret).

For information about creating user accounts, see Managing user accounts on page 145.

For information about how `client_id` and `client_secret` values are used in properties files, see Setting up the Java SDK on page 63 andSetting up the JavaScript SDK on page 67.

3 After the account has been created, log in to the CloudAz console using the account and supply the appropriate `client_secret` password. This is the password specified in the properties file. For security, all users must set their passwords when they log in to their CloudAz console accounts for the first time.

Exception handling

Based on OpenAz specification, `result.allowed()` supports only two responses: `Allow` and `Deny`. If the PDP evaluation does not match the request to any of the system policies, it raises a `PepException`. The suggested behavior for handling this exception is for the PEP to `Deny` the request. This ensures the application is secured by default if there is no policy explicitly allowing access.

Table 5.1: Exception handling

Language	Exception example
Java	`catch(error) { effectValue = "Deny"; }`
JavaScript	`/* * pepResponse.allowed() returns a boolean value. If the * decision is allowed it returns true; otherwise it returns * false for deny * * The above method call throws a PepException for all other * indeterminate states */ String effectValue = ""; try { effectValue = pepResponse.allowed() ? "Allow" : "Deny"; } catch (PepException ex) { effectValue = "Indeterminate"; }`

SDK quick reference

This section provides a quick reference for the OpenAz PEP SDK. Topics include:

- Methods
- Request category classes
- PEPResponse

Methods

Use the following methods when making authorization requests.

Table 5.2: Methods used in authorization requests

Method	Description and example
newInstance	Used to create an instance of a category class, such as Subject.
Java example	`Subject user = Subject.newInstance("chris.webber@hdesk.com");`
addAttribute	Used to add attributes to subject, resource, environment, and recipient category objects.
Java example	`Subject user = Subject.newInstance("chris.webber@hdesk.com");` `user.addAttribute("user_id", "chris.webber")`
JavaScript example	`var user = new Subject("chris.webber@hdesk.com");` `user.addAttribute("department", "IT", "Development");`
simpleDecide	Used to send requests to the server for testing and development. This method takes a user ID, an action, and a resource ID. However, you cannot pass user or resource attributes, discretionary policies, environment details, and so on using this method.
Java example	`PepResponse pepResponse = pepAgent.simpleDecide("Frank Underwood", "OPEN", "Document.docx");`

JavaScript example	`pepAgent.simpleDecide("Frank Underwood", "OPEN", "Document.docx").then(function(pepResponse) {`
	`assert(pepResponse instance of PEPResponse);`
	`});`

decide	Used to send a single authorization request to the server, which expects a dynamic list of categories, and returns a promise that resolves to a single PEPResponse object. This method is more complex than `simpleDecide()` because it requires the construction of user and resource and action objects, rather than IDs alone. With that complexity, however, comes flexibility and power. This method enables you to send information such as user or resource attributes, environment information, destination resource information, host information, and discretionary policies with the request.
Java example	`PepResponse pepResponse = pepAgent.decide(user, action, resource, environment);`

Table 5.2: Methods used in authorization requests (Continued)

Method	Description and example

JavaScript example	`var promise = agent.decide(subject, resource, action, application);`
	
	`promise.then(function(result) {`
	
	`// result is an object of PEPResponse`
	`// TODO process the response`
	`}`

bulkDecide()	Used to send multiple authorization requests to the server in a single call or payload. For example, a web page might have several restricted items. When a user visits the page, the PEP can evaluate the authorization request for all of those items in a single call. This might or might not be more efficient than `decide()`, depending on the number of queries, the speed of the network, the cost of evaluating policies, and so on, but it is conceptually cleaner. For single authorization requests, however, use `decide()`. The server expects the first argument to be an array of objects of the multiple category (called associations), followed by the rest of the category objects. Returns a promise that resolves to an array of `PEPResponse` objects following the order of the associations.

Java example	`List<Resource> resources = new ArrayList<Resource>();`
	`resources.add(resource1);`
	`resources.add(resource2);`
	
	`List<PepResponse> responses = pepAgent.bulkDecide(resources, subject, action, host, application, recipient, environment);`

JavaScript example	`var promise = agent.bulkDecide([resource,resource1], subject, action, application);`
	
	`promise.then(function(result) {`
	`result.forEach(function(r, index) {`
	
	`// r is an object of PEPResponse`
	`// TODO process response`
	`});`
	`});`

Request category classes

Request category classes represent components in an authorization request, such as the subject, action, and resource. Instances of such classes may be passed to the `decide` and `bulkDecide` methods of the `PEPAgent`. The `simpleDecide` method cannot be used to pass subject or resource attributes.

Request category classes include:

- Subclasses of the `org.apache.openaz.pepapi.CategoryContainer`
  - `org.apache.openaz.pepapi.Subject`
  - `org.apache.openaz.pepapi.Action`
  - `org.apache.openaz.pepapi.Resource`

    - `org.apache.openaz.pepapi.Environment`

- NextLabs specific category classes include:
    - `com.nextlabs.openaz.pepapi.Application`
    - `com.nextlabs.openaz.pepapi.Host`
    - `com.nextlabs.openaz.pepapi.Recipient`

Table 5.3: Request category classes

Category class & code example	Description	Data type/ valid input	Required/optional
Subject	The individual performing the action. Typically, an a SID (Windows security identifier or UNIX ID) or an email address.	String	Required
Java example	`Subject user = Subject.newInstance("chris.webber@hdesk.com");` `user.addAttribute("user_id", "chris.webber");`		

JavaScript example	`var Subject = nextlabsopenaz.Subject;`
	`var user = new Subject("chris.webber@hdesk.com");`
	`user.addAttribute("user_id", "chris.webber");`		

Action	The short name of the action, such as edit, view, or copy, for which access is requested. The short name must match that of the Action resource type.	String	Required
Java example	`Action action = Action.newInstance("VIEW_TKTS");`		
JavaScript example	`var Action = nextlabsopenaz.Action;` `var action = new Action("VIEW_TKTS");`		
Resource	The document or other resource on which the action is being performed. The following attribute is required: `ID_RESOURCE_RESOURCE_TYPE`. The value of this attribute must match the short name of the resource type.	String	Required;
Java example	`Resource resource = Resource.newInstance("Ticket:1103");` `resource.addAttribute(Constants.ID_RESOURCE_RESOURCE_TYPE.stringVal ue(), "support_tickets");`		

JavaScript example	`var Resource = nextlabsopenaz.Resource;`
	`var resource = new Resource("Ticket:1103");`
	`resource.addAttribute(NextLabsXACML.ID_RESOURCE_RESOURCE_TYPE, "support_tickets")`
	`resource.addAttribute("ticket_id", "1103");`		

Application	The application used to access the resource and perform the actions.	String	Required for JavaScript only; not used in Java
JavaScript example	`var Application = nextlabsopenaz.Application;` `var application = new Application("Payroll App");`		

Table 5.3: Request category classes (Continued)

Category class & code example	Description	Data type/ valid input	Required/optional
Environment	The conditions, such as location or computer operating system, under which the action is performed.	String	Optional
Java example	`Environment environment = Environment.newInstance();` `environment.addAttribute("authentication_type", "multifactor");`		

JavaScript example	`var Environment = nextlabsopenaz.Environment;`
	`var environment = new Environment();`
	`environment.addAttribute("authentication_type", "multifactor");`		

Recipient	A single email address with attributes or a list of email addresses; Used with email enforcers such as NextLabs Outlook Enforcer. Attributes can be associated with a single recipient only. If your policies depend on attributes, and you need to specify multiple recipients, each recipient must be placed in a separate request.	String; email address	Optional

Java example	`Recipient recipient = Recipient.newInstance`
	`("frank.underwood@whitehouse.gov",`
	`"claire.underwood@whitehouse.gov",`
	`"doug.stamper@whitehouse.gov");`
	
	`Recipient recip = Recipient.newInstance`
	`("frank.underwood@whitehouse.gov");`
	`recip.addAttribute("title", "President");`		

JavaScript example	`var Recipient = nextlabsopenaz.Recipient;`
	`var recipient = new Recipient("frank.underwood@whitehouse.gov");`
	`recipient.addAttribute("title", "President");`		

Host	The hostname or IP address of the machine on which the action is being performed.	String or Integer	Optional
Java example	`Host host = Host.newInstance("mymachine.mycompany.com");`		

JavaScript example	`// default localhost`
	`var host = new Host();`
	
	`// using host name`
	`var host1 = new Host("some_host");`
	
	`// using inet address`
	`var host2 = new Host(0x7f000001);`		

PEPResponse

The Cloud PDP responds to authorization requests by returning the `PEPResponse` object. This object includes the evaluation decision and any obligations that the PEP should attempt to perform when it enforces the decision.

Table 5.4: PEPResponse

Response object	Description
allowed	Returns a value representing the authorization decision: • **true**: Returns the decision, Allow. • **false**: Returns the decision, Deny. • **indeterminate** or **invalid**: Returns the decision, `Not Applicable` or `PEPException`. See Exception handling.
getObligations	Returns the obligations (instructions for performing additional tasks) required by the policy being enforced.
Java example	Returns the set of `org.apache.openaz.pepapi.Obligations` associated with the current result indexed by `ObligationId`.

JavaScript example	Returns a map containing the obligations returned by the Cloud PDP:
	`var promise = agent.decide(subject, resource, action, application);`
	`promise.then(function(result) {`
	`  var obligations = result.getObligations();`
	`  obligations.forEach(function(obl, id) {`
	`  // obl is an object of Obligation`
	`  // TODO process obligation`
	`  });`
	`}`

getWrappedResult	Returns the raw object representing the response. This exposes the helper methods to access the evaluation response details.
Java example	`Decision decision = pepResponse.getWrappedResult().getDecision();`
JavaScript example	`var decision = result.getWrappedResult().Decision;`
Obligation	Instructions for performing additional required tasks, which are sent by the PDP (Policy Decision Point) to the PEP (Policy Enforcement Point) along with the authorization decision.
getId	Returns the obligation ID.
Java example	`String obligationId = obligation.getId();`
JavaScript example	`var obligationId = obligation.getId ();`
getAttributeMap	Returns a map of the obligation attribute name, object-value-array pairs, indexed by name, where name is the `AttributeId` and the value is an array of one or more object values of the attribute. An array with a length greater than one indicates a multi-value attribute.

Table 5.4: PEPResponse (Continued)

Response object	Description
Java example	

```
/*
 * The following section prints all the obligations and their
 * attributes if any obligations are available for the
 * authorization request decision
 */
if (!pepResponse.getObligations().isEmpty()) {
     System.out.println("            Obligations: ");
     Iterator<Entry<String, Obligation>> obligationEntryIter
= pepResponse.getObligations().entrySet()
          .iterator();

     while (obligationEntryIter.hasNext()) {
          Entry<String, Obligation> obligationEntry =
obligationEntryIter.next();
          System.out.println("                { " +
obligationEntry.getKey() + " : [");

     /*
      * Print each obligation's attributes as key values
      */
     Iterator<Entry<String, Object[]>> attrMapIter =
obligationEntry.getValue().getAttributeMap().entrySet()
                         .iterator();
     while (attrMapIter.hasNext()) {
          Entry<String, Object[]> attrMapEntry =
attrMapIter.next();
          System.out.print("                    Key: " +
attrMapEntry.getKey() + ", values:  ");
          for (int j = 0; j < attrMapEntry.getValue().length;
j++) {
                         System.out.print("\"" +
attrMapEntry.getValue()[j] + "\"");
               if (j < attrMapEntry.getValue().length - 1) {
                              System.out.print(", ");
               }
          }

          if (attrMapIter.hasNext()) {
               System.out.println(",");
                    }
                  }
     if (obligationEntryIter.hasNext()) {
          System.out.println(", ");
     }
   }

}
```

Table 5.4: PEPResponse (Continued)

Response object	Description

JavaScript example	`var obligations = result.getObligations();`
	
	`obligations.forEach(function(obligation, key) {`
	
	`var attributes = obligation.getAttributeMap();`
	
	`/*`
	`* Print each obligation's attributes as key values`
	`*/`
	`attributes.forEach(function(values, name) {`
	`console.log("  " + name + ": " + values);`
	`});`
	
	`var ticketId = attributes.get('ticket_id')[0];`
	`var assignedTo = attributes.get('assigned_to')[0];`
	`var processedMsg = String(attributes.get('message')[0]).replace("${ticket_id}", ticketId).replace("${assigned_to}", assignedTo);`
	`console.log("  updated message: " + processedMsg);`
	`console.log("  ");`
	`});`

REST API

REST API Reference

This section explains how to access and use the CloudAz REST API.

Topics in this section

About the CloudAz REST API

The CloudAz REST API is a service that enables a PEP (Policy Enforcement Point) to send policy evaluation requests in XACML to the Cloud PDP (Policy Decision Point), and to receive policy decisions. Requests and responses are sent using HTTPS with mutual authentication. Figure 6-1 illustrates the flow of REST API requests and responses.

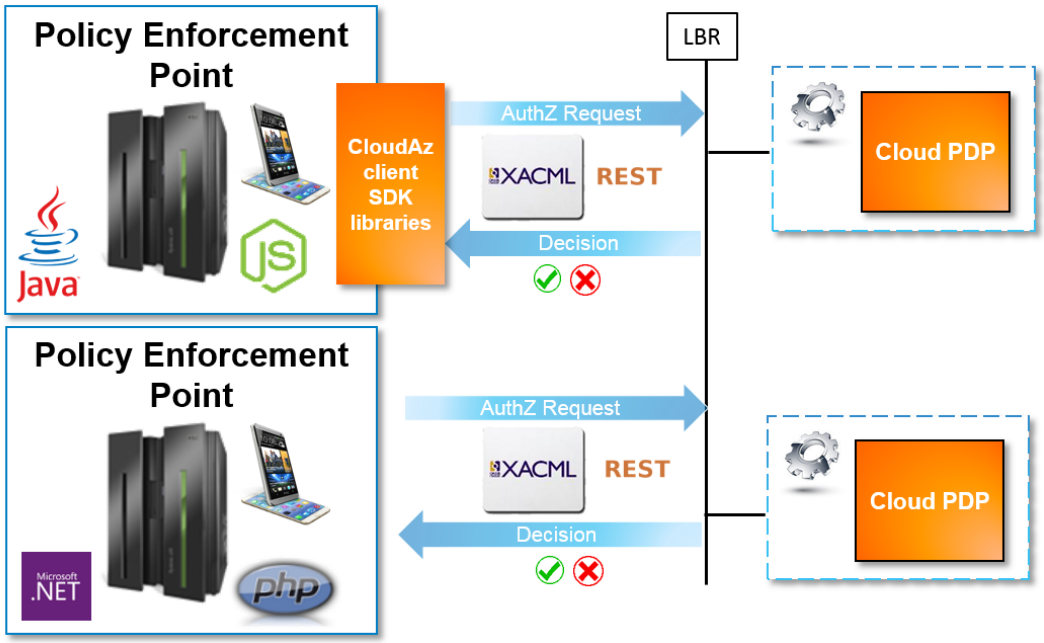


Figure 6-1: Standard message flow using the REST API

How to access the REST API

The REST API URL is provided in the CloudAz *Welcome* email. The URL follows this convention:

`https://<host provided in email>:<SSL-enabled port>/dpc/authorization/pdp`

Using OAuth 2.0 to access Cloud PDP REST APIs

The Cloud PDP protects XACML REST APIs using two-legged OAuth. A Servlet filter verifies that HTTP Headers have valid `Authorization` values before allowing clients to call APIs. `Authorization` values use the OAuth 2.0 `Bearer` token.

Example

```
Authorization : Bearer <access_token>
```

Figure 6-2 shows how the Cloud PDP processes authorization requests.

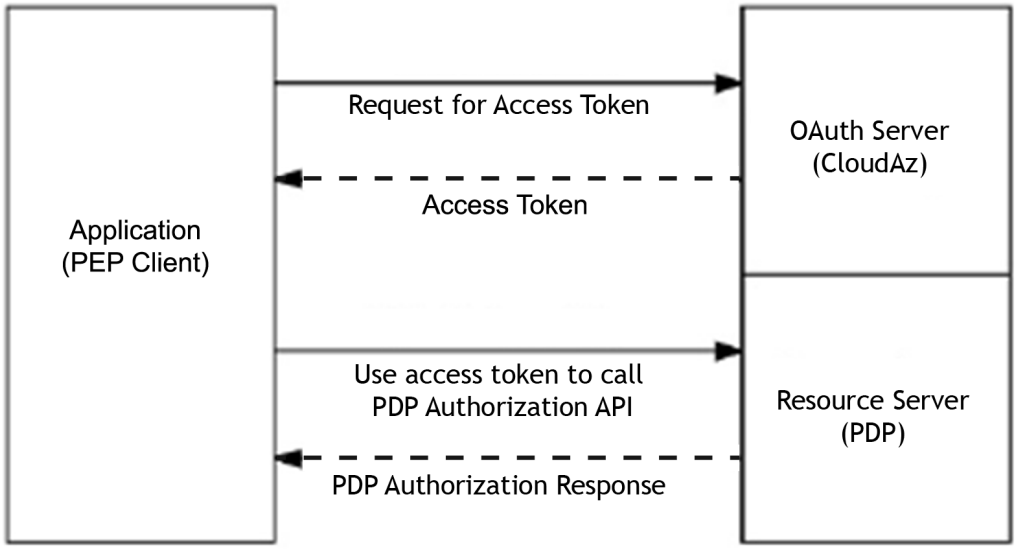


Figure 6-2: How authorization requests are processed

Obtaining an access token

To obtain an access token, make a call to the CloudAz PDP, which functions as the Authorization Server.

About authorization grant types, requests, and responses

CloudAz supports the Client Credentials authorization grant type.

Request endpoint

The following endpoint is used to make an access token request:

```
https://<CloudAz REST Host>/oauth/token
```

Access token request

Clients make requests to token endpoints by adding the following properties using `application/xwww-form-urlencoded` with a character encoding of UTF-8 in the HTTP request entity-body:

- `grant_type` REQUIRED. The value MUST be set to `client_credentials`.
- `client_id` REQUIRED. The API Client ID.
- `client_secret` REQUIRED. The API Client Secret.
- `expires_in` OPTIONAL. The token returned expires in the seconds specified. Users can request a token valid for a maximum of one year, which is 3600 * 24 * 365 seconds. If omitted, the default value, 3600, is used.

Example

```
POST /oauth/token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=apiclient&client_secret=ABCDEFGABCDEFG
```

Access token response

If the access token request is valid and authorized, the authorization server issues an access token.

Example

```
HTTP/1.1 200 OK
{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"Bearer",
    "expires_in":3600,
}
```

Using an access token to access the XACML evaluation service

After obtaining an access token, clients can use that token to call the Cloud PDP XACML evaluation API.

Example

```
POST /dpc/authorization/pdp HTTP/1.1
Host: server.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
... Ohter headers and payload
```

Request format

The REST API supports policy evaluation requests that conform to XACML 3.0 and are passed in XML or JSON format. Requests are sent using the POST method of HTTPS. The format is:

```
Service="EVAL"
DataType=<xml or json>
Data=<XACML request data in JSON or XML>
Version="1.0"
```

The XACML request data passed to the Cloud PDP contains event details, such as the subject (user), action, resource, and application. See Types of requests and examples of request data.

For information about the XACML 3.0 standard, go to http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf.

Types of requests

PEPs can send the following types of policy evaluation requests to the Cloud PDP:

- **Simple requests:** A single policy evaluation query in a single request. A simple request contains a subject and its attributes, a resource and its attributes, and the action the user has performed.
- **Multi-requests:** Multiple policy evaluation queries in a single request. Multi-requests evaluate an action that a user has performed on *multiple* resources.
- **SAML requests**: The use of the SAML (Security Assertion Markup Language) data format to extend the XACML protocol schema. SAML requests enable PEPs to submit authorization request context in a SAML request, along with other information. The PDP uses the attributes passed as SAML assertions.

Types of requests and examples of request data

PEPs can send the following types of policy evaluation requests to CloudAz:

- **Single policy evaluation query in a single request**: Evaluate an action that a user has performed on a document resource. See Single-query request (JSON): Example 1 on page 87 and Single-query request (XML): Example 2 on page 89.
- **Multiple policy evaluation queries in a single request**: Evaluate actions that users have performed on *multiple* document resources. See Multi-query request (XML): Example 3 on page 90.
- **SAML requests**: Exchange authentication and authorization data between identity providers and service providers. The following example shows a SAML request. See SAML request: Example 4 on page 93.

Single-query request (JSON): Example 1

Sample request payload: Example 1

This request data contains an action (`EDIT_TKTS`) that a subject (`chris.webber`) has performed in an application (`Helpdesk`).

```
{
"Request": {
         "ReturnPolicyIdList": "true",
         "Category": [{
              "CategoryId": "urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject",
              "Attribute": [{
                   "AttributeId": "urn:oasis:names:tc:xacml:1.0:subject:subject-id",
                   "Value": "chris.webber@hdesk.com",
                   "IncludeInResult": "false"
              }, {
                   "AttributeId":
"urn:oasis:names:tc:xacml:1.0:subject:assigned_prod_area",
                   "Value": "Exchange Email",
                   "IncludeInResult": "false"
              }, {
                   "AttributeId": "urn:oasis:names:tc:xacml:1.0:subject:department",
                   "Value": "IT",
                   "IncludeInResult": "false"
              }]
         }, {
              "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-
category:resource",
              "Attribute": [{
                   "DataType": "http://www.w3.org/2001/XMLSchema#anyURI",
                   "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-
id",
                   "Value": "Ticket:1103",
                   "IncludeInResult": "false"
              }, {
                   "DataType": "http://www.w3.org/2001/XMLSchema#anyURI",
```

```
                    "AttributeId": "urn:nextlabs:names:evalsvc:1.0:resource:resource-
type",
                    "Value": "support_tickets",
                    "IncludeInResult": "false"
               }, {
                    "DataType": "http://www.w3.org/2001/XMLSchema#anyURI",
                    "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:prod_name",
                    "Value": "Exchange Email",
                    "IncludeInResult": "false"
               }]
          }, {
               "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-
category:action",
               "Attribute": [{
                    "DataType": "http://www.w3.org/2001/XMLSchema#string",
                    "AttributeId": "urn:oasis:names:tc:xacml:1.0:action:action-id",
                    "Value": "EDIT_TKTS",
                    "IncludeInResult": "false"
               }]
          }, {
               "CategoryId": "urn:nextlabs:names:evalsvc:1.0:attribute-
category:application",
               "Attribute": [{
                    "DataType": "http://www.w3.org/2001/XMLSchema#string",
                    "AttributeId":
"urn:nextlabs:names:evalsvc:1.0:application:application-id",
                    "Value": "Helpdesk",
                    "IncludeInResult": "false"
               }]
          }, {
               "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-
category:environment",
               "Attribute": [{
                    "DataType": "http://www.w3.org/2001/XMLSchema#dateTime",
                    "AttributeId":
"urn:oasis:names:tc:xacml:1.0:environment:authentication_type",
                    "Value": "multifactor",
                    "IncludeInResult": "false"
               }]
          }]
     }
}
```

Sample response payload: Example 1

The response allows the action.

```
{
    "Response": {
         "Result": [{
               "Decision": "Permit",
               "Status": {
                    "StatusMessage": "success",
                    "StatusCode": {
                      "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
                    }
               },
               "Obligations": []
```

```
          }]
    }
}
```

Single-query request (XML): Example 2

Sample request payload: Example 2

This request data contains an action (`VIEW_TKTS`) that a subject (`Chris Webber`) has performed on a document resource (`Ticket:1103`).

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 http://
docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
ReturnPolicyIdList="false">
   <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject">
      <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">chris.webber@hdesk.com</AttributeValue>
      </Attribute>
      <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:department">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">IT</
AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
      <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#String">Ticket:1103</AttributeValue>
      </Attribute>
      <Attribute IncludeInResult="false"
AttributeId="urn:nextlabs:names:evalsvc:1.0:resource:resource-type">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#String">support_tickets</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
      <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">VIEW_TKTS</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes Category="urn:nextlabs:names:evalsvc:1.0:attribute-
category:application">
      <Attribute IncludeInResult="false"
AttributeId="urn:nextlabs:names:evalsvc:1.0:application:application-id">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#String">MSExcel</
AttributeValue>
      </Attribute>
   </Attributes>
```

```
   <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:environment">
      <Attribute AttributeId="authentication_type" Issuer=""
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">multifactor</AttributeValue>
      </Attribute>
   </Attributes>
</Request>
```

Sample response payload: Example 2

The response allows the action.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <Result>
        <Decision>Permit</Decision>
        <Status>
            <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
            <StatusMessage>success</StatusMessage>
        </Status>
        <Obligations/>
    </Result>
</Response>
```

Multi-query request (XML): Example 3

Sample request payload: Example 3

This request contains two subjects (`chris.webber@hdesk.com` and `amy.tan@hdesk.com`) who have performed two different actions (`EDIT_TKTS` and `ASSIGN_TKTS`) respectively on the support tickets resource (`Ticket:1103`), using the `Helpdesk` application.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
   <Attributes xml:id="subject1" Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject">
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">chris.webber@hdesk.com</AttributeValue>
      </Attribute>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:department"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">IT</
AttributeValue>
      </Attribute>
     <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:assigned_prod_area"
IncludeInResult="false">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Exchange
Email</AttributeValue>
      </Attribute>
```

```
   </Attributes>
   <Attributes xml:id="subject2" Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject">
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">amy.tan@hdesk.com</AttributeValue>
      </Attribute>
      <Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:assigned_prod_area"
IncludeInResult="false">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Exchange
Email</AttributeValue>
      </Attribute>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:department"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">IT</
AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes xml:id="action1" Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">EDIT_TKTS</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes xml:id="action2" Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">ASSIGN_TKTS</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes xml:id="application1"
Category="urn:nextlabs:names:evalsvc:1.0:attribute-category:application">
      <Attribute
AttributeId="urn:nextlabs:names:evalsvc:1.0:application:application-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Helpdesk</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes xml:id="resource1"  Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource">
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">Ticket:1103</AttributeValue>
      </Attribute>
     <Attribute AttributeId="urn:nextlabs:names:evalsvc:1.0:resource:resource-type"
IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">support_tickets</AttributeValue>
      </Attribute>
     <Attribute AttributeId="urn:nextlabs:names:evalsvc:1.0:resource:resource-
dimension" IncludeInResult="false">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">from</
AttributeValue>
      </Attribute>
```

```
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:prod_name"
IncludeInResult="false">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Exchange
Email</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes xml:id="environment1"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
        <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:authentication_type" >
             <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#dateTime">multifactor</AttributeValue>
        </Attribute>
    </Attributes>
   <MultiRequests>
     <RequestReference>
       <AttributesReference ReferenceId="subject1"/>
       <AttributesReference ReferenceId="resource1"/>
       <AttributesReference ReferenceId="action1"/>
       <AttributesReference ReferenceId="application1"/>
       <AttributesReference ReferenceId="environment1"/>
     </RequestReference>
     <RequestReference>
       <AttributesReference ReferenceId="subject2"/>
       <AttributesReference ReferenceId="resource1"/>
       <AttributesReference ReferenceId="action2"/>
       <AttributesReference ReferenceId="application1"/>
       <AttributesReference ReferenceId="environment1"/>
     </RequestReference>
  </MultiRequests>
</Request>
```

Sample response payload: Example 3

The response allows the action.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
   <Result>
      <Decision>Permit</Decision>
      <Status>
         <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok" />
         <StatusMessage>success</StatusMessage>
      </Status>
      <Obligations />
   </Result>
   <Result>
      <Decision>Permit</Decision>
      <Status>
         <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok" />
         <StatusMessage>success</StatusMessage>
      </Status>
      <Obligations />
   </Result>
</Response>
```

SAML request: Example 4

SAML is used to exchange authentication and authorization data between identity providers and service providers. The following example shows a SAML request.

```
<xacml-samlp:XACMLAuthzDecisionQueryType InputContextOnly="true" IssueInstant="2011-
10-31T06:44:57.766Z" ReturnContext="false" Version="2.0" xmlns:xacml-
samlp="urn:oasis:names:tc:xacml:2.0:profile:saml2.0:v2:schema:protocol">
    <saml:Issuer SPProvidedID="SPPProvierId"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">https://
identity.carbon.wso2.org</saml:Issuer>
   <xacml-context:Request xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
       <xacml-context:Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
           <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://
www.w3.org/2001/XMLSchema#string">
               <xacml-context:AttributeValue>chris.webber@hdesk.com</xacml-
context:AttributeValue>
           </xacml-context:Attribute>
           <xacml-context:Attribute AttributeId="department" DataType="http://
www.w3.org/2001/XMLSchema#string">
               <xacml-context:AttributeValue>IT</xacml-context:AttributeValue>
           </xacml-context:Attribute>
       </xacml-context:Subject>
       <xacml-context:Resource xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
           <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://
www.w3.org/2001/XMLSchema#string">
               <xacml-context:AttributeValue>Ticket:1103</xacml-
context:AttributeValue>
           </xacml-context:Attribute>
           <xacml-context:Attribute
AttributeId="urn:nextlabs:names:evalsvc:1.0:resource:resource-type" DataType="http:/
/www.w3.org/2001/XMLSchema#string">
               <xacml-context:AttributeValue>support_tickets</xacml-
context:AttributeValue>
           </xacml-context:Attribute>
       </xacml-context:Resource>
       <xacml-context:Action xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
            <xacml-context:Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://
www.w3.org/2001/XMLSchema#string">
              <xacml-context:AttributeValue>VIEW_TKTS</xacml-context:AttributeValue>
           </xacml-context:Attribute>
       </xacml-context:Action>
       <xacml-context:Environment xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"/>
   </xacml-context:Request>
</xacml-samlp:XACMLAuthzDecisionQueryType>
```

NextLabs-specific environment attributes

NextLabs specific-environment attributes can be used to specify what happens when authorization requests do not match deployed policies or when error conditions occur. There are two ways to use these environment attributes:

- **Including attributes in authorization requests sent by the PEP**: Environment attributes included in authorization requests are applied only to the current request.
- **Setting attributes as system properties of the PDP**. Environment attributes that are configured in the PDP properties are applied to every request processed by the PDP.

There are two NextLabs-specific environment attributes:

- dont-care-acceptable
- error-result-acceptable

dont-care-acceptable

Used to specify what happens when authorization requests do not match deployed policies. When this attribute is set to yes or true, the PDP returns **Indeterminate** when the request does not match any of the deployed policies.

To use `dont-care-acceptable` in an authorization request, pass the attribute with the value `yes`.

Example of the attribute sent in a request

```
   {
   "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-category:environment",
   "Attribute": [
   { "DataType": "http://www.w3.org/2001/XMLSchema#string", "AttributeId":
"urn:oasis:names:tc:xacml:1.0:environment:dont-care-acceptable", "Value": "yes",
"IncludeInResult": "false" }
   ]
   }
```

To use `dont-care-acceptable` as a system variable, include the attribute with the value `true` as a JVM system variable to be applied at PDP startup. The PDP returns indeterminate for every request that does not match a policy.

Example of the attribute as a system variable

```
   -Dnextlabs.dont.care.acceptable=true
```

error-result-acceptable

Used to specify what happens when errors are encountered. If this attribute is set to yes or true, the PDP returns `Error` if it encounters an error while processing the request.

To use `error-result-acceptable` in a request, pass attribute with the value `yes`.

Example of the attribute set in a request

```
   {
   "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-category:environment",
   "Attribute": [
   { "DataType": "http://www.w3.org/2001/XMLSchema#string", "AttributeId":
"urn:oasis:names:tc:xacml:1.0:environment:error-result-acceptable", "Value": "yes",
"IncludeInResult": "false" }
   ]
   }
```

To use `error-result-acceptable` as a PDP environment variable, include the attribute with the value `true` as a Java system variable during the PDP startup. The attribute is applied globally, and the PDP returns `Error` whenever it encounters an error while processing any request.

Example of the attribute as a system variable

```
   -Dnextlabs.error.result.acceptable=true
```

Product Guide

Part 1 Working with Policies

1.1 Policy and Policy Model overview

This section provides an overview of policies and explains how to create and manage Policy Model resource types and policy components.

Topics in this section

About policies

Policies are statements that describe resource usage situations and specify the actions to be taken when those situations arise. In other words, policies define a set of rules controlling how categories of users in a given environment are allowed to use categories of resources. Policy designers construct policies as combinations of components that are linked with operators and other logical constraints, and then further refined by contextual conditions, such as time of day.

Typically, organizations construct all the policies required to cover all the potential business situations where information must be controlled, or where an event, such as displaying a reminder message, should be triggered in response to a user action. Each policy comprises a set of predefined building blocks that are combined according to the syntax shown in Figure 1.1-1.

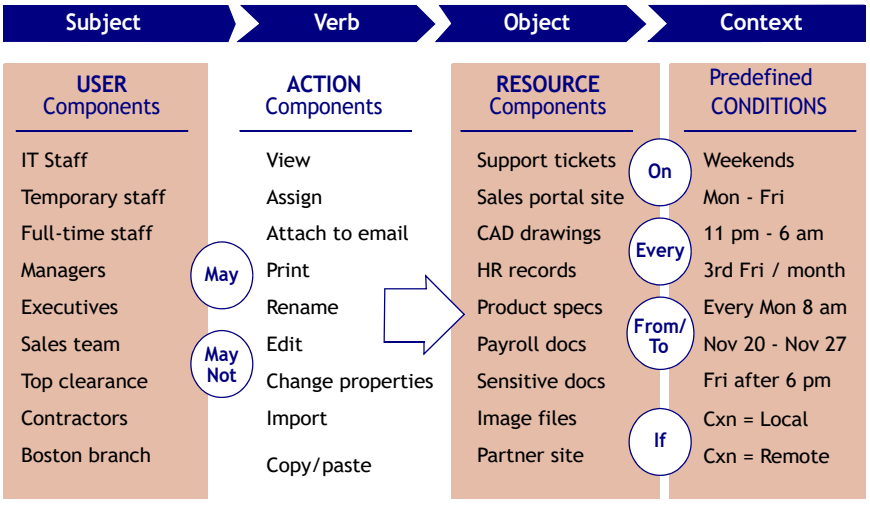


Figure 1.1-1: Policy components and syntax

Overview of policy implementation

Implementing policies involves these tasks:

- **Creating a Policy Model**: The Policy Model determines the kind of subject types (users) and resource types (documents and other resources) that can be covered by policies. To create a Policy Model, you create resource types, and specify the attributes, actions, and obligations available to each resource type during policy creation. See Managing Policy Model resource types.
- **Defining policy components**: Policy components represent categories or classes of entities, such as users, and actions, such as open or copy. These components can be thought of as the parts of speech used to construct policy statements. For example, "noun" policy components might include *All employees in the IT department, Support tickets,* or *Any file with an XLS extension*, and "verb" components might include *Edit, Copy*, *Print*, or *Rename File*. See Managing policy components.
- **Using components to construct policies:** Policies use components as building blocks to represent rules that control information access and use. For example, the components in the following policy, "Allow Users in IT to Edit and Reassign Tickets in their Product Area," are "IT Department," "Edit," "Assign," and "Tickets in User's Product Area." Policies are stored in a Policy Master database for distribution to the network components that enforce them. *Conditions* are policy elements that change a policy's effect based on dynamic comparisons, evaluations, or contextual factors. Complex conditions can be written directly in ACPL (Active Control Policy Language), the language CloudAz uses internally to represent, store, and manage components and policies. Policies can also include obligations, such as displaying notifications to end users and sending email messages to administrators when policies are enforced. See Constructing and testing policies.
- **Deploying policies to the Policy Decision Points (PDPs):** Deploying policies means distributing policies to the appropriate CloudAz PDPs. Policies are enforced only after they are deployed. See Deploying and managing objects.
- **Fine-tuning policies after they have been deployed**: After policies have been deployed, they can be monitored and updated as needed to ensure that they are performing as required. See Managing policies and components.

Managing Policy Model resource types

Policy Model resource types are the templates that include information about attributes, actions, and obligations available to components and policies.

Adding and editing resource types

You can add or edit resource types at any time. To make attributes, actions, and obligations available to policy designers, resource types should be added before components are created.

Procedure

1 Log in to CloudAz with an account that has permission to add resource types.

2 On the left navigation bar, click **Policy Model**.

3 Do one of the following:

- Click **Add Resource Type**.
- Click the name of an existing resource type to edit it.

4 Provide *Resource Type Information*. This information is displayed on the *Policy Models* page and identifies the resource type in the system:

Table 1.1.1: Resource Type Information

Option	Description
Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?
Short Name	The identifier used to reference an item in the database. Underscores (_) are allowed, but short names cannot contain spaces or special characters, cannot be changed, and must be unique in the system. Developers should make a record of short names because short names are required for API calls. For example, to reference attributes in a request, you would use this format: resource.pat_rec.age, where pat_rec is the short name for Patient Record, and age is the short name for Age.
Description	A description of the item. This field is useful for explaining how the item is used and for documenting changes to the item. Descriptions can include multibyte characters, such as those used by Asian languages.

Table 1.1.1: Resource Type Information (Continued)

Option	Description
Tags	Labels that identify or classify or group items according to business needs. You can apply tags to items such as policies, components, and resource types.This classification has these main benefits: • It simplifies search operations. • It simplifies the management of policies (delegated administration). • It enables you to create reports based on tags in the Reporter console. For example, to allow only citizens of the USA to manage ABAC policies related to export control, you could add a tag to each policy that enforces export controls such as ITAR. You could then define a delegation policy that allows access to administrators whose citizenship is USA. Each item can have any number of tags. For example, if a policy restricts access to data in classified information, the following tags might be useful: • Top Seccret • Secret • Confidential Although there are many other attributes associated with policies—such as user name, resource attributes, action, and policy name—on which you can filter in Reporter, tags are often the most useful because you can add your own key words and values that make the items easier to identify and find. **Important**: Before creating tags for policies, define the reporting requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention and apply tags uniformly to ensure that policy activity reports that are filtered by tags return the results you expect. For example, if some policies governing export control data contain the tag, Export Control = ITAR, while some export control policies contain the tag, Classification = ITAR, and others do not contain any tags, filtering on one tag does not capture all the export control policy activity data. Similarly, define the delegation requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention, and apply tags uniformly to ensure that data are filtered by tags to return the results you expect. **Note**: Tags used in components are available only to components; component tags are not available to policies, and policy tags are not available to components. Tags cannot include these characters: ~ / * $ & \ ? { }

5 Click the **Attributes** tab, then add the attributes that you want to make available to policy designers who select the resource type:

Table 1-2: Attribute information

Option	Description
Attribute Name	The display name of the item. Names cannot be longer than 128 characters, and cannot include the reserved characters *$*, */*, *?*, \, *&*, *or* *. Names can include multibyte characters, such as those used by Asian languages.
Short Name	The internal identifier associated with the item. The system uses short names to reference items in the database. Short names cannot contain spaces or special characters, and they cannot be changed after they are added. In addition, short names must be unique in the system. Developers should keep a record of the short names because they are required for API calls.

Table 1-2: Attribute information

Option	Description
Data Type	The kind of information in the attribute. Data types include strings or text, numbers, and collections. Collections are sets of data, such as locations or vendors.
Operators	The symbol selected for the operation. The available operators are determined by the data type selected. *MULTIVALUE* • not equal (!=): The item does not match the attribute multival. • equal (=): The item matches the attribute multival. • exactly matches: The item matches all of the items in the attribute multival. • includes: The item contains one or more items in the attribute collection. *STRING* • is: The item matches the attribute string. • is not: The item does not match the attribute string. *NUMBER* • equals (=): The item matches the attribute number. • not equals (!=): The item does not match the attribute number. • less than (<): The item is less than the attribute number. • less than equals (<=): The item is less than or equal to the attribute number. • greater than (>): The item is greater than the attribute number. • greater than equals (>=): The item is greater than or equal to the attribute number.

6 Click the **Actions** tab, then add the actions that you want to make available to policy designers who select the resource type. Actions are the operations that can be performed on the resource type:

Table 1.1.1: Resource type actions

Option	Description
Action Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?
Short Name	The identifier used to reference an item in the database. Underscores (_) are allowed, but short names cannot contain spaces or special characters, cannot be changed, and must be unique in the system. Developers should make a record of short names because short names are required for API calls. For example, to reference attributes in a request, you would use this format: resource.pat_rec.age, where pat_rec is the short name for Patient Record, and age is the short name for Age.

Note: If you create an action from within a resource type, the system automatically creates a corresponding action component. The new action component has these properties:

- The name of the action component is the name given to the action.
- The description of the action component indicates that it was generated automatically by the system.
- Tags currently assigned to the resource type are automatically assigned to the action component.
- If you delete the action from within the resource type deletes the action component provided that the component is not being used elsewhere. Components can be deleted only if they are not in use.

7 Click the **Obligations** tab, then add the obligations that you want to make available to the resource type:

Note: Obligations are additional instructions the PDP might return to the PEP together with the authorization decision. For example, an obligation might instruct the PEP to notify the user why they are not allowed to perform a certain operation, and the message itself might be included in the obligation. In other cases, obligations might instruct the PEP to record the allowed user action in an audit database; encrypt a value before allowing the user to store that value; or inject a security filter in a SQL query, to restrict the rows being returned, before allowing the user to run the query against a given database table.

Table 1.1.2: Resource type obligations

Option	Description
Obligation Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?
Short Name	The identifier used to reference an item in the database. Underscores (_) are allowed, but short names cannot contain spaces or special characters, cannot be changed, and must be unique in the system. Developers should make a record of short names because short names are required for API calls. For example, to reference attributes in a request, you would use this format: resource.pat_rec.age, where pat_rec is the short name for Patient Record, and age is the short name for Age.

Table 1.1.2: Resource type obligations (Continued)

Option	Description

Parameters	Information used to prepopulate fields in obligations. For example, if the obligation is to send email, an obligation parameter could include an email address. This simplifies the policy authoring experience by prepopulating values.
	Parameters information includes:
	• **Name**: The name of the parameter.
	• **Short Name**. The identifier used to reference the parameter in the database. Underscores (_) are allowed, but short names cannot contain spaces or special characters, cannot be changed, and must be unique in the system. Developers should make a record of short names because short names are required for API calls. For example, to reference attributes in a request, you would use this format: resource.pat_rec.age, where pat_rec is the short name for Patient Record, and age is the short name for Age.
	• **Type**: The category of the parameter. Categories have differing properties:
	  • Single Row: A single value.
	  • Multiple Row: Items that appear in multiple rows.
	  • List: A range of values.
	• **List value**: For List type parameters, the values to be displayed in the list. Separate each value with a comma.
	• **Default Value**: The value that appears by default.The default value might be an instruction, such as <Enter the reason why access is denied> to be replaced during policy component creation. Or the default value might include variables and expressions. See Using Advanced Conditions.
	• **Hidden**: Whether or not the parameter is displayed during policy component creation.
	• **Editable**: Whether or not the parameter can be edited during policy component creation.
	• **Mandatory**: Whether or not the parameter must be specified during policy creation.

8 Click **Save** at the upper right of the page. The resource type is added to the Policy Model. It is available for selection during policy component creation.

Next steps

Create additional resource types, or add policy components. See Managing policy components on page 107.

Managing policy components

Policy components are the abstract building blocks or raw material of information control policies. They represent categories and classes, either of physical entities that play a role in information control policies, or of actions that involve those entities. Because components are categories and classes rather than physical entities, they provide a layer of abstraction that insulates an organization's information control policies from changes to the physical entities in the environment.

For example, employees might join or leave the organization, documents might be created and deleted, computers are purchased and retired, and SharePoint sites or pages change on a daily basis. Despite these changes, you can write policies without needing to know specifically about the underlying users, documents, or servers. The policies remain in effect, covering all appropriate network entities, even as those entities change.

About component types

CloudAz supports the following categories of components:

- Resource
- Action
- Subject

You create components by specifying a component type and defining the component's properties. After you save and deploy components, you can use them to construct policies that govern not some *specific* user in your organization, but *any* user belonging to the category that the component describes.

For example, rather than listing every IT department user in your organization by name, you could create a user-type component that represents the category, "all users in the IT department." Then you could use that component as a building block to construct policies that control whether users can access and use the specified information. This means that whoever designs policies using policy components does not need any direct knowledge of who the IT department users are, or of who joins or leaves the IT department in the future. They simply need to know how, as a class, IT department users should be allowed to access and use information.

Action components are a combination of one or more of the available basic actions defined in the resource type policy model. Action components are generated automatically when they are added from within the resource type.

For example, in the Support Tickets Policy Model, there are three basic actions:

- EDIT_TKTS
- ASSIGN_TKTS
- VIEW_TKTS

An action component can be created by combining one, two, or three of these actions. Typically, if you want to create a policy for EDIT_TKTS, you would want to grant VIEW_TKTS as well. So instead of creating two policies, you could create one action component that includes both. You can only use action components for policy creation, so even if you are using one basic action, you need to create a component. So to minimize that step, NextLabs pre-creates a component with only one basic action during policy model creation. Administrators can modify, add basic actions, and bundle actions as required.

You can use components in various combinations to construct the policies you need for managing information access and use. For example, you might define a user component "Contractors," and an action component "Copy, Move, Print, or Send via Email or IM." You could then combine these to implement an "eyes-only" rule: "All independent contractors may read company documents but may not transmit them in any way." When this rule is distributed to all CloudAz enforcement

clients in your network, it is enforced regardless of the individual contractor or document involved.

The process of defining components involves constructing a model of component categories that is comprehensive enough to represent all the actual entities in your organization that need to be covered by policies. You can either make a component model based on your knowledge of the network entities, regardless of how they may need to be used in policies; or based on the requirements of a set of logical policies that have already been designed.

Note: You might find it expedient to construct a set of policies first, and then define the components you need for those policies. See About constructing policies.

Adding and editing policy components

You can add policy components to define resources, actions, and subjects in your environment any time as needed. After policy components are defined, they become available to policy designers for use in policies.

Before you begin

Set up the Policy Model by creating resource types for use in components. See Adding and editing resource types.

Procedure

1. Log in to CloudAz with an account that has permission to add components.
2. On the left navigation bar, in the *Components* section, select a component type:
   - Subject
   - Resource
   - Action
3. Do one of the following:
   - Click **Add Component**.
   - Click the name of an existing component to edit it.
4. If you are adding a resource or action component, select the resource type to use as the template for this component. The list is empty if no resource types have been added. See Adding and editing resource types.
5. Provide the component information. This information is displayed on the *Components* page and used to identify the component in the system:

Table 1.1.3: Component information

Option	Description
Display Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?

Table 1.1.3: Component information (Continued)

Option	Description
Description	A description of the item. This field is useful for explaining how the item is used and for documenting changes to the item. Descriptions can include multibyte characters, such as those used by Asian languages.
Tags	Labels that identify or classify or group items according to business needs. You can apply tags to items such as policies, components, and resource types.This classification has these main benefits: • It simplifies search operations. • It simplifies the management of policies (delegated administration). • It enables you to create reports based on tags in the Reporter console. For example, to allow only citizens of the USA to manage ABAC policies related to export control, you could add a tag to each policy that enforces export controls such as ITAR. You could then define a delegation policy that allows access to administrators whose citizenship is USA. Each item can have any number of tags. For example, if a policy restricts access to data in classified information, the following tags might be useful: • Top Seccret • Secret • Confidential Although there are many other attributes associated with policies—such as user name, resource attributes, action, and policy name—on which you can filter in Reporter, tags are often the most useful because you can add your own key words and values that make the items easier to identify and find. **Important**: Before creating tags for policies, define the reporting requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention and apply tags uniformly to ensure that policy activity reports that are filtered by tags return the results you expect. For example, if some policies governing export control data contain the tag, Export Control = ITAR, while some export control policies contain the tag, Classification = ITAR, and others do not contain any tags, filtering on one tag does not capture all the export control policy activity data. Similarly, define the delegation requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention, and apply tags uniformly to ensure that data are filtered by tags to return the results you expect. **Note**: Tags used in components are available only to components; component tags are not available to policies, and policy tags are not available to components. Tags cannot include these characters: ~ / * $ & \ ? { }

6 Do any of the following:

- Select the conditions to be used with the component. Conditions appear only if they have been added to the selected resource type.
- Select the Actions to be available with the component. Actions appear only if they have been added to the selected resource type.

- Select or create subcomponents to include in the component. Subcomponents are components that are included in the parent component's hierarchy. For example, you could have a component called *Residents of the USA*, and include subcomponents such as *Residents of California*, *Residents of Virginia*, and so on.

  To create subcomponent, type a name in the *Include Subcomponents* field, add conditions for the new subcomponent, then click **Apply**.

  Note: You can specify as many subcomponents as you need for each component. The conditions in multiple subcomponents are evaluated with OR operators.

7 Do one of the following:

- Click **Save** to save the component in the *Draft* state.
- Click **Save & Deploy** to make the component available for deployment with policies.

Next steps

Create additional components or subcomponents, or construct policies that use the components. See About constructing policies.

Note: If you chose *Save*, you must deploy the component before policies that use it can be enforced. See Deploying components.

1.2 Constructing and testing policies

This section describes how to construct policies. Policies are rules that are designed to control access to resources in an organization. Policies consist of components combined with logical operators, variables, obligations, and time-based contexts.

Topics in this section

About constructing policies

Policy designers construct policies to control access to resources in an organization. For example, a designer might construct the following policy:

> Allow Users in IT to Edit and Reassign Tickets in their Product Area

The components used in this policy include:

- Subject component: IT Department (User)
- Action components: Edit and Assign (Support Tickets)
- Resource component: Tickets in User's Product Area (Support Tickets)

Policy designers use conditions to change a policy's effect based on dynamic comparisons, evaluations, or contextual factors. In addition, policies can include notification obligations, such as messages to end users or email notifications to administrators when policies are enforced anywhere in the network.

Adding policies

Before you begin

Configure a Policy Model and configure policy components. See Policy and Policy Model overview.

Procedure

1 Log in to CloudAz with an account that has permission to add policies.

2 On the left navigation bar, click **Policies.**

3 Click **Add Policy**.

4 Provide the policy information. This is used to identify the policy on the *Policies* list and in the CloudAz system:

Table 1.2.1: Policy information

Option	Description
Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?
Description	A description of the item. This field is useful for explaining how the item is used and for documenting changes to the item. Descriptions can include multibyte characters, such as those used by Asian languages.
Tags	Labels that identify or classify or group items according to business needs. You can apply tags to items such as policies, components, and resource types.This classification has these main benefits: • It simplifies search operations. • It simplifies the management of policies (delegated administration). • It enables you to create reports based on tags in the Reporter console. For example, to allow only citizens of the USA to manage ABAC policies related to export control, you could add a tag to each policy that enforces export controls such as ITAR. You could then define a delegation policy that allows access to administrators whose citizenship is USA. Each item can have any number of tags. For example, if a policy restricts access to data in classified information, the following tags might be useful: • Top Seccret • Secret • Confidential Although there are many other attributes associated with policies—such as user name, resource attributes, action, and policy name—on which you can filter in Reporter, tags are often the most useful because you can add your own key words and values that make the items easier to identify and find. **Important**: Before creating tags for policies, define the reporting requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention and apply tags uniformly to ensure that policy activity reports that are filtered by tags return the results you expect. For example, if some policies governing export control data contain the tag, Export Control = ITAR, while some export control policies contain the tag, Classification = ITAR, and others do not contain any tags, filtering on one tag does not capture all the export control policy activity data. Similarly, define the delegation requirements, and then identify the tags needed to support those requirements. Use a consistent naming convention, and apply tags uniformly to ensure that data are filtered by tags to return the results you expect. **Note**: Tags used in components are available only to components; component tags are not available to policies, and policy tags are not available to components. Tags cannot include these characters: ~ / * $ & \ ? { } **Note:** Tags used in policies are available only to policies; policy tags are not available to components, and component tags are not available to policies.

5 Select the policy effect and subject components:

Table 1.2.2: Policy effect and subject components

Option	Description
Policy Effect	The action taken when conditions in the policy are met. • **Allow**: Permit the listed Subjects to perform the task specified in the policy. Allowing a set of Subjects to perform a task does not mean that others are blocked from performing that action. Allow policies can never result in Deny responses. Allow policies can only result in Allow or Not Applicable responses. • **Deny**: Do not permit the listed Subjects to perform the task specified in the policy, but allow all others to do so. Deny policies can result in Deny, Allow, or Not Applicable responses.
Subject Components (Add Condition)	The subjects (users) to be governed by the policy. Conditions are expressions of predicates that refine the context in which policy actions are performed. Conditions can be selected only if attributes have been added for the subject type policy component. See Adding and editing policy components. Using conditions, you can create policies that change their effect based on dynamic comparisons, evaluations, or contextual factors. For complex contextual factors, use Advanced Conditions, which are written directly in ACPL (Active Control Policy Language). Click the plus icon next to the condition that you want to set. See Using Advanced Conditions. **Important**: If no subject components are specified, the policy applies to all subject components.

6 Select resource components. These are the information resources, such as documents, to be governed by the policy. Do any of the following:

- To select from all existing resource components, click in the field, then select a resource component.
- To search for a resource component, begin typing in the field, then select a resource component.
- To add a resource component, type a name, select the check box next to the name, which is identified as *(NEW)*, then click **Apply**. An empty resource component is added to the policy. The new resource component also appears on the *Resource Components* list. Edit the resource component to define its properties. See Adding and editing policy components.
- To add multiple resource components, click **Add Condition**, then select or add a resource component.

Note: If you add multiple components in the same Subject component, they are evaluated with the OR logical operator. To perform an AND logical operation, add a condition.

- To remove a resource component, click the **X** next to the resource component name.

7 Select action components. These are the actions to be governed by the policy. If you include more than one action, an OR operator is assumed. If no action components are specified, the policy applies to all action components.

8 **Optional**: In the Advanced Conditions section, add any ACPL (Active Control Policy Language) expressions you want to use in policy evaluation. This is useful for creating conditions that cannot be defined using resource and user components. For example, if you have a secure application, you could write an ACPL expression that includes multifactor authentication:

```
environment.authentication_type = "multifactor"
```

For more information on Advanced Conditions, see Using Advanced Conditions.

9 Choose the policy effective duration. This is the period of time when the policy is to be enforced. Setting a duration is useful if you have a policy that you want to enforce only during a particular time period. For example, allowing users to access resources during working hours only, or for the duration of a specific project:

Option	Description
Always	Enforce the policy continuously.
Specific Days	Enforce the policy according to the specified schedule: • Specify a date range to enforce the policy during a specified period. • Specify the days of the week on which the policy is to be enforced. For example, select SUN and SAT to enforce the policy on weekends only (Sunday and Saturday). Time is evaluated according to the CloudAz Policy Management server's time zone, which matches the service region specified in the CloudAz subscription. Trial subscriptions use Pacific time.

Note: It is important to understand the difference between these time condition settings. If a user does something to a resource and it is within this specified time period, the policy is enforced; if it is outside that period, the policy is not enforced. However, the policy is evaluated in either case.

10 Select the obligations you want the CloudAz PDP (Policy Decision Point) to pass to the PEP (Policy Enforcement Point) when an authorization request matches a policy.

11 Do one of the following:

- Click **Save**.
- Click **Save & Deploy**.

The policy is added to the Policy Master database.

Next steps

If you chose *Save*, you must deploy the policy to distribute it to enforcers. Policies are enforced only after they are deployed. See Deploying and managing objects.

Using Advanced Conditions

Advanced Conditions are ACPL (Active Control Policy Language) expressions that can be used to include parameters that cannot be conveyed by policy components, such as user and resource components. These conditions can be used to change a policy's effect based on relationships between resource, subject, and host attributes that are dynamically returned on a policy query.

For instance, a condition might compare the attribute values returned for a resource (the Product Line a document is associated with) with the attribute values returned for a subject (the Product Line a user is associated with). The syntax for this condition would be:

```
resource.<resource type>.<attribute> = subject.<attribute>
```

The Advanced Condition would be:

```
resource.ProductLine = user.ProductLine
```

In addition, policy designers can combine multiple attribute-to-attribute match conditions, as well as other matches (attribute-to-string, attribute-to-constant, attribute-to-integer). Matches can also be defined using different methods, listed in Table 1.2.3. Advanced Conditions use Boolean logic, so all Match Operators can be reversed to "Not." Where multiple matches are combined, they can be linked using a logical "AND" or "OR". Finally, partial and full wildcards can be used to replace all or parts of strings. Examples with explanations are shown in Table 1.2.3.

Table 1.2.3: Match operators for Advanced Conditions

Name	ACPL Operator	Description
Equal or Multi-value equal (match any)	=	When a single value is present for each attribute, A = B when A and B values match. When multiple values are present for attributes, {A} = {B} when any value in set {A} matches any value in set {B}.
Multi-value unordered equal	equals_unordered	When multiple values are present for each attribute, {A} *equals_unordered* {B} when set {A} and set {B} are identical, where order is not relevant.
Set includes	includes	Where multiple values are present for attributes, {A} *includes* {B} when all values in set {B} are present in set {A}. This can also apply where {B} is an empty set. Where multiple values are present for set {A} and a single value is present for B, set {A} includes {B} when the single value in {B} is present in set {A}.
Greater than/equal to, less than/equal to	>, >=, <, <=	A matches B, where the value returned for attribute A is greater than; greater than or equal to; less than; or less than or equal to the value of B.

Using wildcards in Advanced Conditions

You can use wildcard characters in attribute values. Generally they are useful to compare URL strings, filenames, and other attributes with patterns. Table 1.2.4 shows supported wildcards.

Table 1.2.4: Wildcards used in Advanced Conditions

Wildcard character	Description
*	Matches zero or more characters, not including the path separator "/"
**	Matches zero or more characters, including the path separator "/"
d	Matches any single-digit numeral, 0-9
D	Matches any numeral, one or more digits
a	Matches any single letter, either upper- or lowercase
A	Matches one or more letters, either upper- or lowercase
c	Matches any single alphanumeric character or special character, such as %, &, _, etc., except for backslashes, which are reserved
C	Matches one or more alphanumeric characters or special characters, such as %, &, _, etc., except for backslashes, which are reserved
s	Matches a single space
s	Matches one or more spaces

With the exception of the standard `*` and `**`, all wildcards consist of a letter preceded by an escape character. Two escape characters are supported, a question mark and an exclamation point:

- `?<wildcard>` means *Matches the wildcard*. This is useful for general purposes, such as matching policies to existing resources.
- `!<wildcard>` means *Does Not Match the wildcard*. Beyond general matching purposes, this is useful to ensure that new filenames conform to required conventions.

If the character following the `?` or `!` is not one of the wildcard characters listed above, it is assumed to represent itself. Thus `?z` is just the letter `z`. More usefully, `??` is a literal question-mark and `?!` is a literal exclamation point.

Different letters represent different kinds of characters, and each has two versions: lowercase stands for a single character of a certain type, and uppercase stands for one or more of those characters. This wildcard convention enables you to define wildcard-based conditions with a great deal of precision. For example:

- `http://www.mycompany.com\*.html` matches any html file at the top level of http://www.mycompany.com
- `http://www.mycompany.com/**/*.html` matches any html file anywhere in the URL
- `**.txt` matches any resource ending in ".txt"
- `**/secret*` matches any resource with a name that starts with "secret"

- `**/*secret*` matches any resource with a name that contains the word “secret” anywhere.
- `Q?drevenue.doc` matches any resource with a name that starts with “Q”, ends with “revenue.doc”, and has exactly one single-digit number in between. For example, \\documents\accounting\Q2revenue.doc would match, while \\documents\accounting\Q4-98revenue.doc would not match.
- `?c?c?c.*` matches any resource with a name containing exactly three characters of any kind, in either upper- or lowercase.

Example condition for equal and multi-value equal

The following condition includes an attribute-to-attribute multi-value equal AND an attribute-to-string match.

```
(resource.fso.ProductLine = user.ProductLine) AND (resource.fso.ReviewStatus =
"Approved")
```

In this example, the user attribute ProductLine must match the resource attribute ProductLine, while multiple attributes are possible for both. As noted in Table 1.2.3, for a multi-value match, the “=” operator evaluates to true when *any* of the values match. In other words, if a document belongs to ProductLine A, B, and C, and the user belongs to ProductLine C, D, and E, the condition is met. The attribute-to-string match specifies that the resource attribute ReviewStatus must match the string “Approved.” Because a logical AND links the two conditions, both must match for the entire condition to be true.

Example condition for multi-value equals_unordered operators

The “equals_unordered” operator should be used in cases where all attribute values must match between two multi-value sets, where the order of values is not relevant. A typical use case is for a policy that allows access only when a document is not export controlled, OR the user requesting access to the resource has ALL the proper export licenses associated with that resource.

```
(resource.fso.ExportControl="NO") OR (resource.fso.ExportLicense
equals_unordered user.ExportLicense)
```

Example condition for the includes operator

The “includes” operator should be used in cases where set {A} should be included within set {B}. This is different from “equals_unordered,” as set {B} might have additional values not present in set {A}. This is also different from “=” as set {B} can be an empty set.

A possible use case for this operator could be where a user must have all the appropriate class (or security) attributes as a resource to be granted access to that resource. In other words, user set {A} must include resource set {B}, where the user has all the proper clearances associated with the resource. Notice that this use case is different than equal or multi-value equal, as set {B} can be empty. If the resource has no security settings associated with it, user set {A} can include the empty resource set {B}, and the user should be able to access the resource.

```
resource.fso.DocSecurity includes user.SecurityAccess
```

Example condition for greater than/less than

In addition to equal matches, you can also use greater/less than, and greater/less than or equal to operators in Advanced Condition. Some examples could include policies that match dynamic attributes to dates ("date last accessed" or "today's date," or the date a license or policy is still valid), or integers (number of times accessed).

```
resource.fso.LicenseEndDate >= current_time
```

Managing subpolicies

Subpolicies are policies that can be included in other policies; they are the children of parent policies. Subpolicies are typically used to specify additional policy enforcement criteria in the form of exceptions. For example, if a parent policy specifies that access to sensitive data is denied to all, a subpolicy can specify an exception to that rule. The subpolicy might state that users with special clearance can access the sensitive data. Subpolicies are especially useful for policies that include or exclude many classes of subjects. Separating these exceptions into subpolicies makes it easier to read and manage policies.

Examples of subpolicies

The following examples, expressed in English, show typical ways to use subpolicies.

In the following example, a parent policy denies access to confidential resources. Three subpolicies specify exceptions to the parent policy, allowing three types of users—licensed users, executives, and members of the legal department—to access confidential resources.

```
Deny Access to Confidential Resources
     Allow Licensed Users
     Allow Executives
     Allow Legal Department
```

The next example shows a hierarchical set of policies. The top-level policy denies access to sensitive financial documents. The first subpolicy specifies an exception to the top-level policy: Members of the finance group are allowed access. The second subpolicy specifies an exception to the first subpolicy: contractors in the finance group are denied access. The third subpolicy specifies an exception to the second subpolicy: contractors in the finance group with special clearance are allowed access.

```
Deny Access to Sensitive Financial Documents
     Allow Finance Group
          Deny Contractors
               Allow Contractors with Special Clearance
```

As the examples show, subpolicies provide an intuitive way to express complex policies that specify multiple conditions or exceptions. The examples also show the best way to design policies: Start with a policy that applies broadly, and then enumerate specific exceptions to that general

policy. This is an important design concept because it enables you to define policies that are modular and extensible.

Adding subpolicies

You can add subpolicies for any policy.

Procedure

1. On the left navigation bar, click **Policies**.
2. Locate the policy to which you want to add a subpolicy.
3. In the policy row, open the *Action* menu on the right, then select the **Add Subpolicy** button.
4. Define the properties of the subpolicy. Because a subpolicy is simply a policy within a policy, you define and deploy a subpolicy in the same way you define a parent policy. See Adding policies.

   When the subpolicy has been added, it appears on the policy list. It also appears in the *Policy hierarchy* section of the parent policy.

Viewing and editing policy and component information

You can view information related to policies and components, including the hierarchy among related policies or components, their change history, and their properties.

Viewing policy and component hierarchy

You can view the relationships between policies and components in the policy hierarchy. This is especially useful for policies that have subpolicies and subcomponents.

Procedure

1. On the left navigation bar, do one of the following:
   - Click **Policies.**
   - Click a component type.
2. Click the name of the policy or component whose hierarchy you want to view.
3. In the left pane, click the hierarchy button: 

   Note: The hierarchy button is available only for policies and components that have been saved.

Viewing policy and component version history

You can view the version history of policies and components in the policy properties. This information includes the date and time of changes and the name of the user who made the changes.

Procedure

1 On the left navigation bar, do one of the following:

- Click **Policies.**
- Click a component type.

2 Click the name of the policy or component whose history you want to view.

3 In the left pane, click the history button: 

Note: The history button is available only for policies and components that have been saved.

Viewing and editing policy and component properties

You can view and edit policy and component properties as needed.

Procedure

1 To view or edit policy or subpolicy properties:

a On the left navigation bar, select **Policies**, then click the name of the policy or subpolicy you want to view or edit.

b View or edit the policy properties as needed, then click **Save**.

2 To view or edit component properties:

a On the left navigation bar, select the type of component whose properties you want to view or edit, then click the name of the component.

b View or edit the component properties as needed, then click **Save**.

Cloning policies and components

When you need to create policies and components that are similar to existing policies and components, it is often more efficient to make a copy of existing items and then make the desired changes in the duplicate, rather than build the new policy and components from scratch.

Cloning policies

Duplicating policies is referred to as *policy cloning*. When you clone a policy, any subpolicies within the policy are also cloned.

Procedure

1. On the left navigation bar, click **Policies**.
2. Locate the policy you want to clone.
3. In the policy row, open the *Action* menu on the right, then select the **Clone** button. The cloned policy is saved in *Draft* form and appears on the policy list.
4. Edit the cloned policy as needed. See Adding policies.

Note: The components used by cloned policies are *not* themselves cloned; rather, the cloned policy and the source policy share the original components. To change the definition of one or more components as well as the policy, you need to clone the components as well. See Cloning components.

Cloning components

Duplicating components is referred to as *component cloning*. All types of components can be cloned.

Procedure

1. In the `Components` section on the left navigation bar, select a component type, then locate the component you want to clone.
2. In the component row, open the *Action* menu on the right, then select the **Clone** button. The cloned component is saved in *Draft* form and appears on the component list.
3. Edit the cloned component as needed. See Managing policy components.
4. Click **Save** to clone the selected components.

   If the component is based on any other components (these are called *leaf components*), those components are not cloned. Rather, the cloned copy and the source component share the original leaves. To change the definition of the leaf components, you need to clone the leaf components as well, using the same procedure.

Constructing Delegated Administration policies

Delegated Administration policies enable administrators to manage user access to the CloudAz consoles as well as policies, and policy objects using rules based on user attributes and conditions. This mirrors the way policies are used to control access to resources and eliminates the need to create and apply roles to users or groups of users. See Adding and editing delegation policies.

Testing policies

After policies are saved, you can test them using the CloudAz console.Testing policies in the console simulates sending authorization requests and enables you to view responses without building PEPs (policy enforcement points) or setting up application interactions. This reduces testing time and enables you to:

- Construct authorization requests by selecting attributes.
- View example request and response payloads in JSON and XML formats.
- Verify that policies function as expected. For example, you can verify that PEPs pass all of the necessary attributes and actions in requests, and that PDPs (Policy Decision Points) provide the appropriate responses.
- Select multiple policies for testing to ensure that policies function correctly in combination. For example, one policy might allow users with the attribute *Suppliers* to access a customer list. Another policy might deny access to *Suppliers* unless their territory matches the territory of the customer list. Testing both policies at once enables you to verify that the policies perform as expected when combined.

Note: To test policies, you must use an account with permission to view the Policy Tester, policies, and Policy Model components.

Also, policies that are in the *Inactive* state, that is, policies that have previously been deployed but that are currently deactivated, cannot be tested until they are deployed again.

Before you begin

Before you begin, you must be familiar with the policies you want to test and the attributes specified in those policies. You need to select those attributes to generate authorization requests.

Procedure

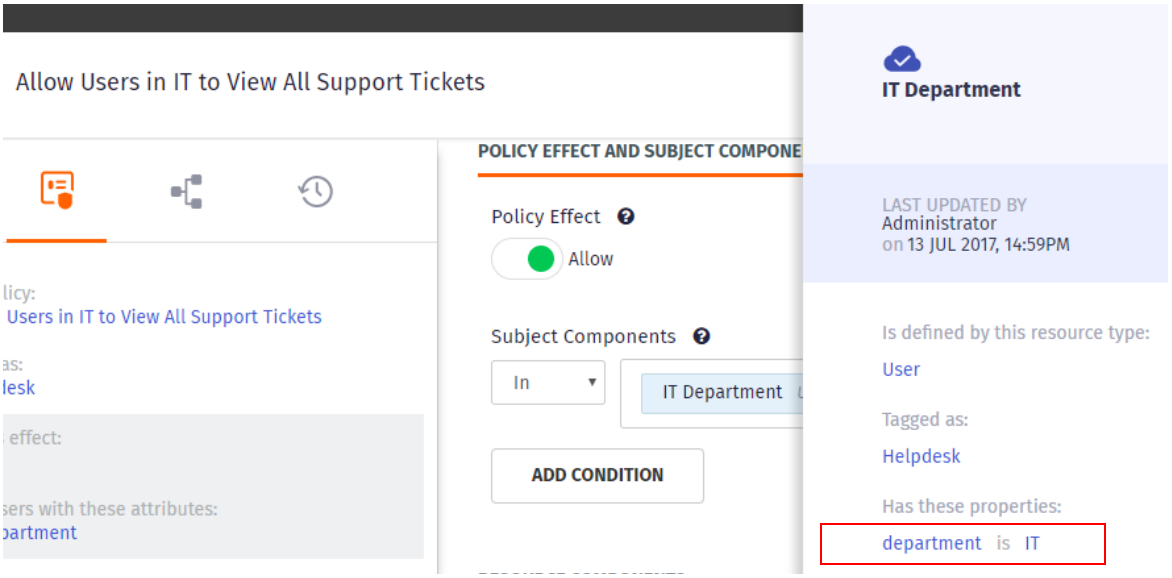
1 On the left navigation bar, click **Policy Tester**.

2 In the *Select Policy* section, choose the policies to be tested:

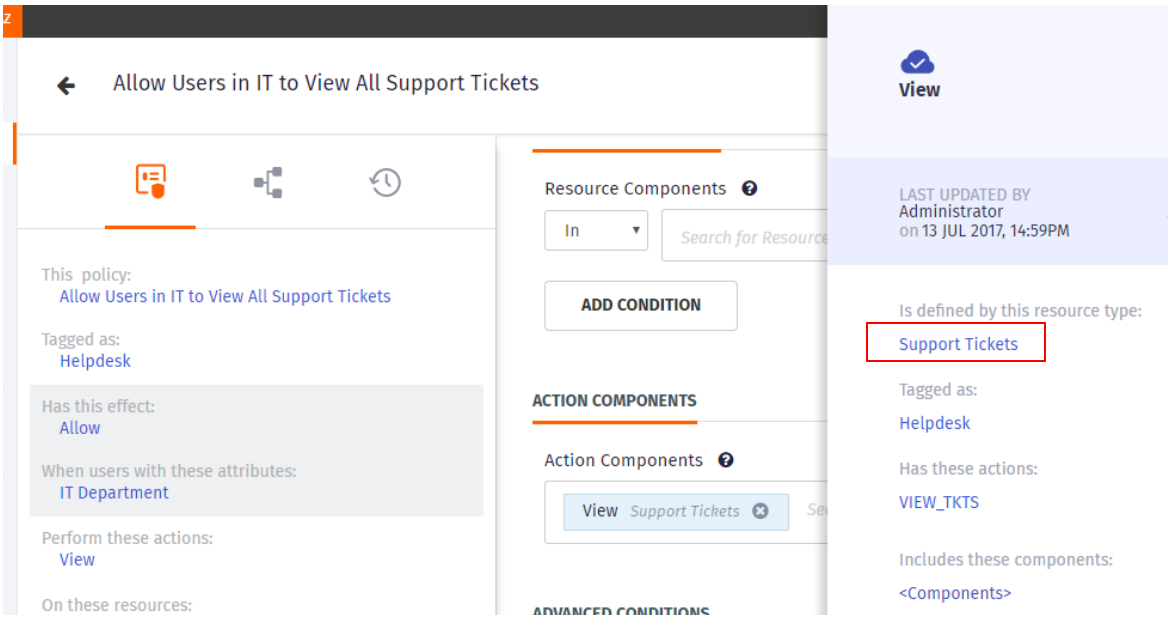
- **All Policies**: All policies you have access to, provided they are in the *Inactive Draft*, *Deployed*, or *Deployed Draft* state. Policies that are in the *Inactive* state, that is, policies that have previously been deployed but that are currently deactivated, cannot be tested until they are deployed again. Select this option to test for conflicts among all policies.
- **Select Policies**: Only the policies specified in this section. After you choose **Select Policies**, type the name of the policies you want to test to search for matching policies.
  - **Include Sub-Policies**: If you choose Select Policies, choose whether to include sub-policies when searching for the policies you want to select.
- **Saved Search Filter**: If you have searched for policies on the *Policy Management page*, and you have saved a search, you can select it to include all the policies in the search results.

3 In the *Request* section, add the attributes you want to include in the test. For the first test, select attributes that match the policy.

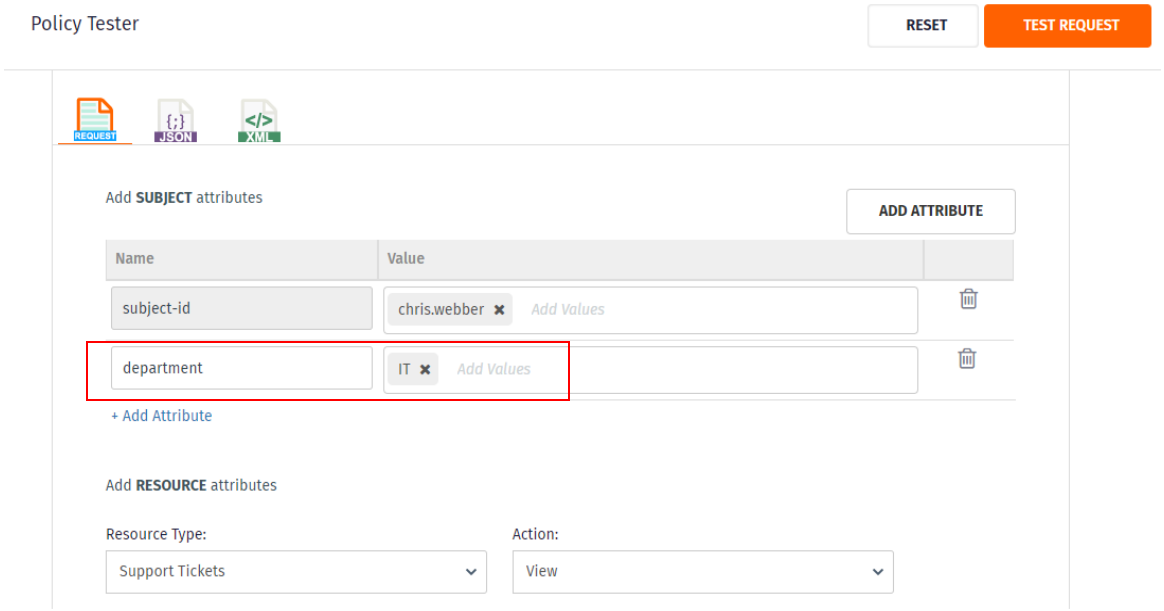
For example, *Allow Users in IT to View All Support Tickets*, a sample policy included in the QuickStart Guide, has the Subject attribute *IT Department*.



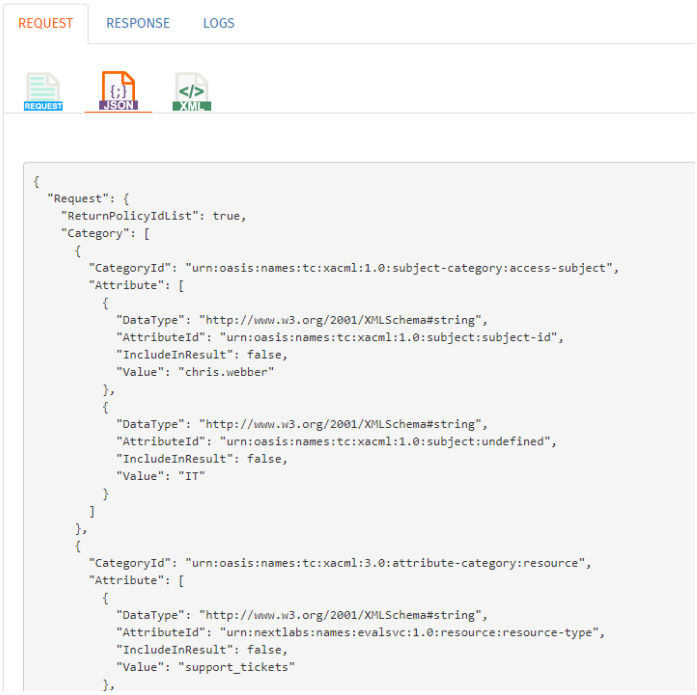
And it includes the action *View*, which is defined by the resource type *Support Tickets*.



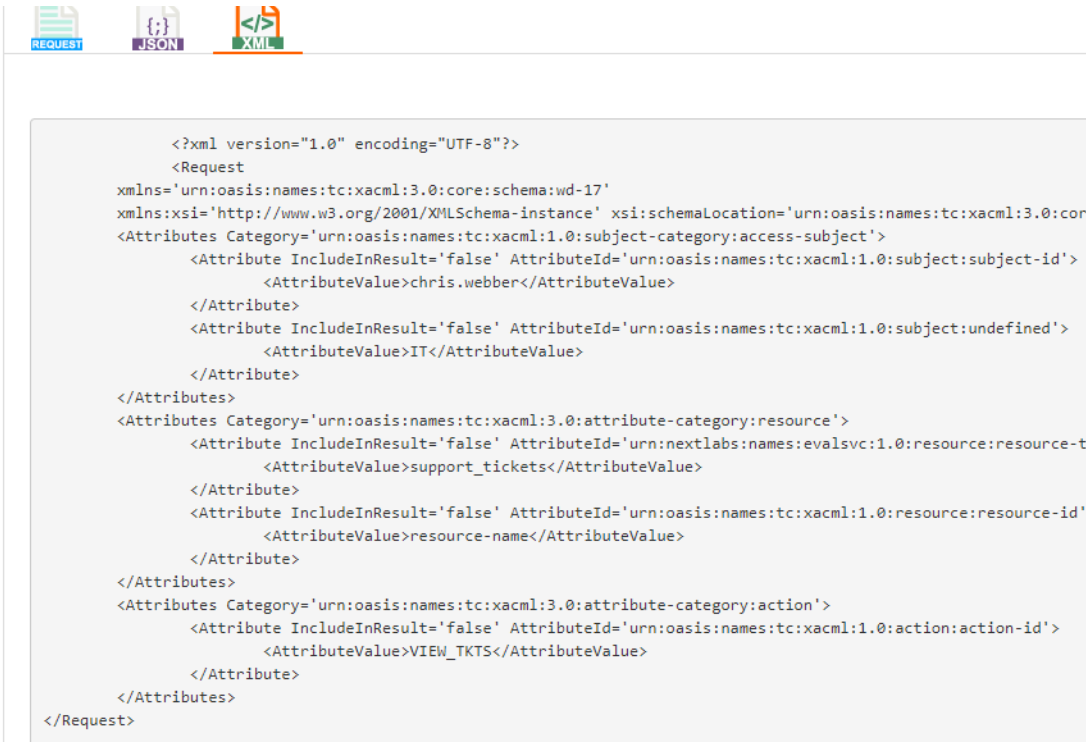
To verify that the *Allow Users in IT to View All Support Tickets* policy works as expected, add the Subject attribute, *Department*, with the value *IT*.



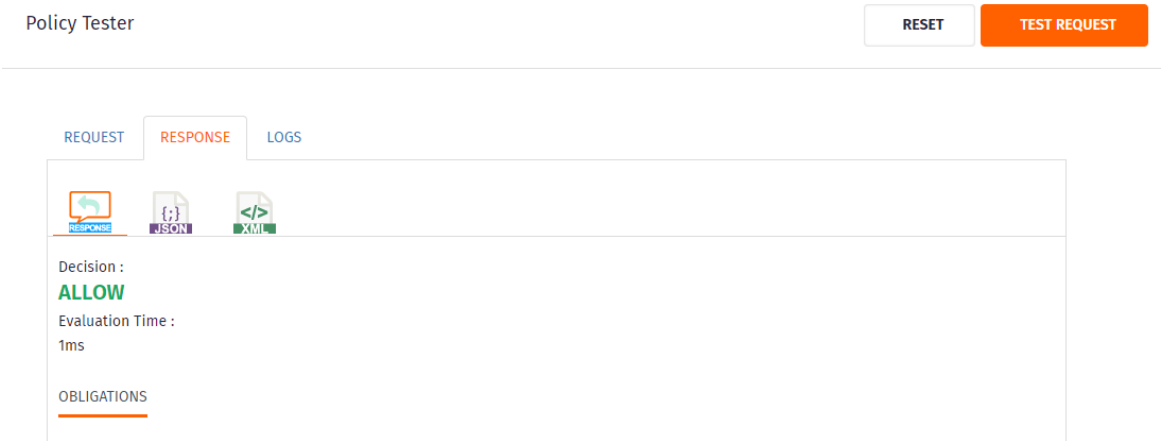
4 To view the JSON build request, click the JSON button under the **Request** tab.



5 To view the XML build request, click the JSON button under the **Request** tab.

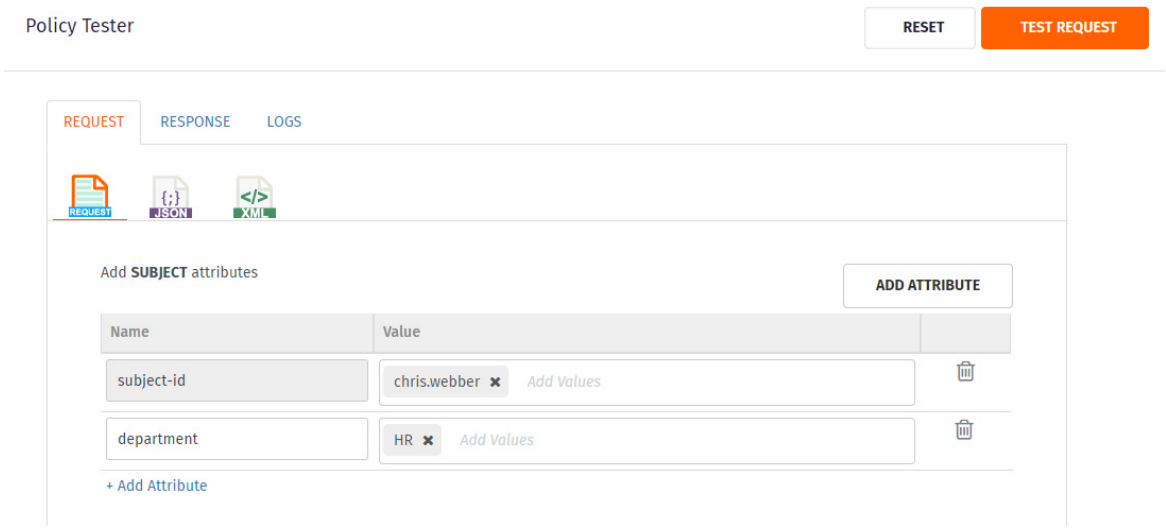


6 To see the PDP response, click **Test Request**.

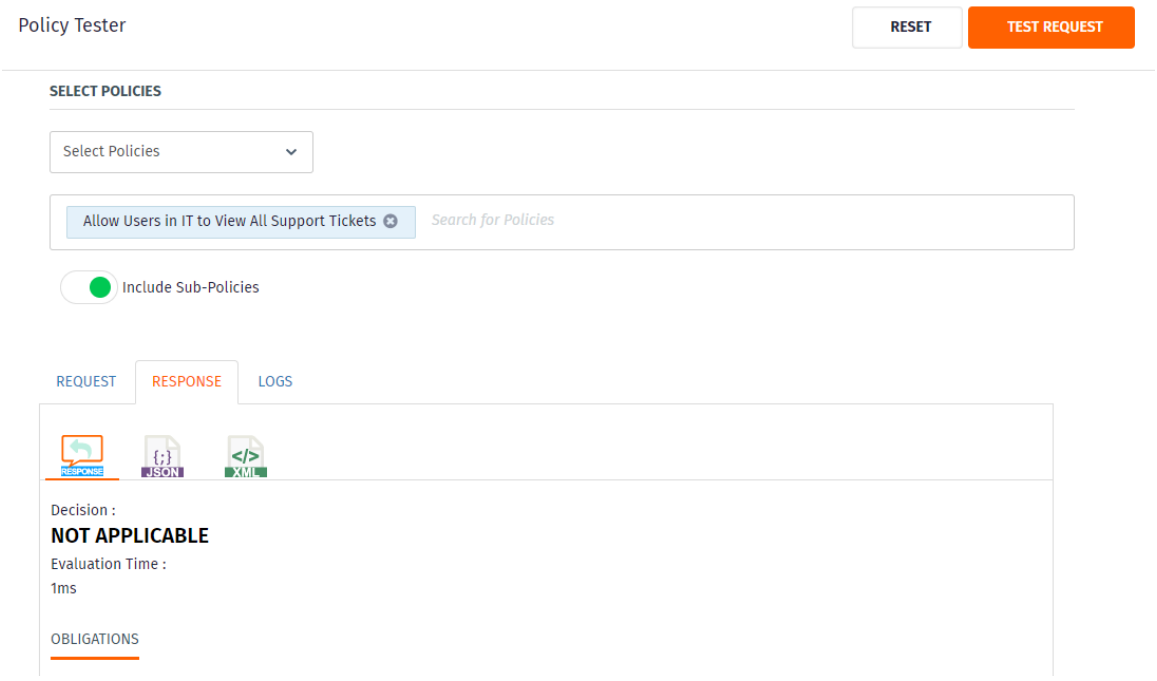


Note: The *Evaluation Time* is the amount of time the PDP took to make the decision. The *Obligations* section shows obligations required by the policy.

7 For another test, change the value of an attribute so that it does not match the policy. In the *Allow Users in IT to View All Support Tickets* example, change the value of the *Department* attribute to something other than *IT*, such as *HR*.

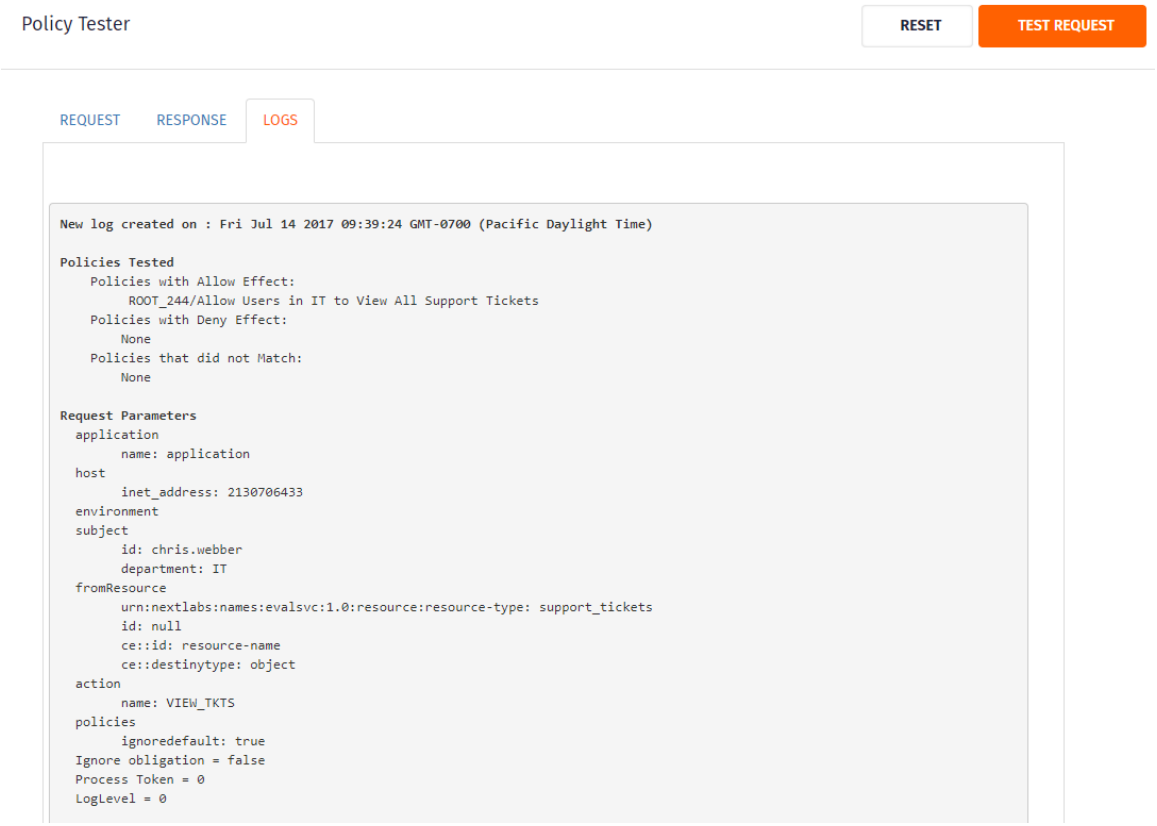


8 To view the response, click **Test Request**.



Note: In production environments, NextLabs recommends that you deny access to resources when the response is *Not Applicable*.

9 For information about how the policy was applied, click the Logs tab.



10 To clear the policy selection and results, click **Reset**.

Note: Test results do not appear in the CloudAz Activity Log, and results are cleared when you refresh the browser.

1.3 Exporting and importing policies

This section explains how to export and import policies among CloudAz instances. In addition, this section describes how CloudAz manages policy versions and states.

Topics in this section

About exporting and importing policies and other items

Exporting and importing policies enables you to share policy objects defined in one instance of with other instances of CloudAz. This is helpful in collaborative environments where policy construction responsibilities are distributed among several policy designers. It is also useful for transferring policies and components from isolated test environments to live production environments.

Policies, and any components used in policies, are exported as BIN files, which can be imported into other instances.

Important: The following items cannot be exported or imported:

- Individual components, such as subject, resource, and action components. Components can be exported and imported only if they are included in policies.
- Policy model components, such as resource types. These can be exported and imported only if they are included in policies.
- Delegation policies. Delegation policies cannot be imported or exported and must be created in each instance of CloudAz separately as needed.

Exporting and importing objects using the CloudAz console

Policies and their related components can be exported and imported using the CloudAz console.

Before you begin

To avoid creating inconsistent policy sets, follow these guidelines:

- Perform exports or imports at a time when you are certain no one is using policy instances.

- Request that all policy analysts log out of the CloudAz console when any policies are being imported.
- Publish the list of policies and components to be imported, and ask all policy analysts to refrain from editing any object in that list until the procedure is complete.

Procedure

1 On the left navigation bar, click **Policies**.

2 On the Policy Management page, do any of the following:

- Select the export button in the upper right to export all policy objects.
- Select the import button in the upper right to import policy objects.
- Select the policy objects you want to export from the list, then click the export button above the list.

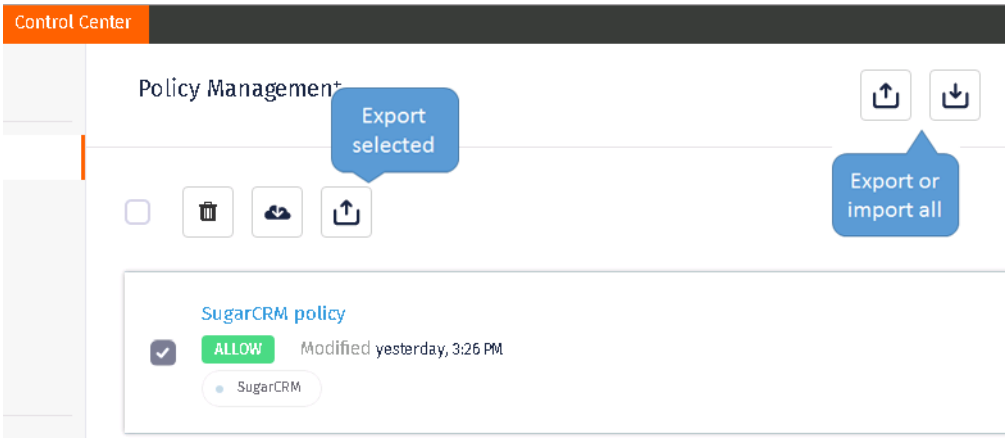


Figure 1.3-1: Export and import controls

When policies are exported, all subpolicies, components, and associated policy model components, such as resource types, are exported to a file with the extension BIN.

When policies are imported:

- All subpolicies, components, and associated policy model components, such as resource types, are imported.
- With the exception of the User subject type, all policies or components whose component names or short names match those being imported are overwritten. For example, if the imported file contains a resource type named `Support_Ticket` and a resource type with the same short name already exists in the environment, the existing resource type is overwritten with the imported version.
- When the User subject type is imported, all new attributes are added, but the existing User subject type attributes are not overwritten or removed.

Using versions

The version mechanism is useful in keeping track of changing objects whose definitions evolve, so you can see the definitions of previously deployed policies. This is most useful for auditing purposes, in cases when you need to demonstrate that a certain set of policy controls were in effect at a certain time.

Each newly created object, in Draft state, starts as Version 0. Once it is deployed, it becomes Version 1. Thereafter, every time the object moves out of Deployed state and is then redeployed, its version increments by one. The version number increments simply by virtue of changing back into Deployed state, regardless of whether any changes have been made to the definition of the object. A date stamp is permanently assigned to each incremental revision, which provides a useful tool for managing and tracking versions.

When working with versions, it is important to understand the distinction between an object's *state* and its *status*. Different versions of the same object can have different statuses, and in such cases, the same object is listed in more than one row in the Object Grid. In addition, if an object has undergone several changes, the different versions of it are listed in the grid, with the same status—for example, all obsolete versions would be listed on the *Deactivated* tab, showing *Inactive* status.

About object states

Table 1.3.1 lists the possible states of components, policies, and Policy Model objects.

Table 1.3.1: Object states

State	Description
Draft	Open for editing. The object has been created and saved, but has not yet been submitted for deployment, or has been taken out of the deployed state for editing.
Deployed	The object has been deployed, and its definition has been sent out to all relevant enforcers.
Deployed Draft	The object has been taken out of the deployed state for editing. An earlier version is deployed.
Inactive	The object has been deactivated, and the deactivated version has been deployed to enforcers.

1.4 Deploying and managing objects

This section explains how to deploy objects, such as policies and policy components. Policies are enforced only after they are deployed.

Topics in this section

About deploying and managing objects

Deployment places objects into a state where policies can be sent to Policy Decision Points (PDPs) at the next heartbeat and enforced throughout the network.

Note: Policies are enforced only after they are deployed.

Deploying components

Components used in policies must be deployed before policies that use those components can be enforced.

Procedure

1. On the left navigation bar of CloudAz, select **Components**, then select a component type.
2. On the component list, do one of the following:
   - Locate the component you want to deploy, then select the deploy button in the *Action* menu at the right of the component row.
   - Select the check boxes next to the components you want to deploy, then click the deploy button on the left above the list.

   The components are deployed and available for enforcement in policies.

Deploying policies

Policies must be deployed before they can be enforced.

1 On the left navigation bar of the CloudAz console, select **Policies**.

2 On the Policies list, do one of the following:

- Locate the policy you want to deploy, then select the deploy button in the *Action* menu at the right of the policy row.
- Select the check boxes next to the policies you want to deploy, then click the deploy button on the left above the list.

CloudAz automatically performs a dependency check to see whether any components used in the policy are in the draft state and must be deployed before the policy is deployed.

3 If dependencies are identified, verify that all components used in the policy are deployed.

Managing policies and components

Policy management tasks include:

- Searching and filtering policy and component lists
- Modifying policies and components
- Changing user access to components
- De-activating policies and components

Searching and filtering policy and component lists

CloudAz enables you to search for policies and components, which is especially useful for finding objects quickly when you have long lists of policies and components.

In addition, search criteria can be used as filters. When search criteria are applied to a list, the system displays only those items that match the criteria.

Procedure

1 Log in to CloudAz with an account that has permission to edit policies or components.

2 To search for and filter policies:

a On the left navigation bar, click **Policies**, then click the magnifying glass button in the upper right of the page.

b Select search criteria, then click **Apply**. The page shows policies that match the criteria. When you close the search panel, the list continues to be filtered according to the search criteria and Filters Applied appears at the top of the list.

c If a filter is applied, click the **X** next to *Filters Applied* to remove it and display all list items.

3 To search for and filter components:

a Select a component type on the left navigation bar, then click the magnifying glass button in the upper right of the page.

b Select search criteria, then click **Apply**. The page shows policies that match the criteria. When you close the search panel, the list continues to be filtered according to the search criteria and Filters Applied appears at the top of the list.

c If a filter is applied, click the **X** next to *Filters Applied* to remove it and display all list items.

4 To search for Policy Model components (resource types and subject types):

a On the left navigation bar, click **Policy Model**, then click the magnifying glass button in the upper right of the page.

b Select search criteria, then click **Apply**. The page shows policies that match the criteria. When you close the search panel, the list continues to be filtered according to the search criteria and Filters Applied appears at the top of the list.

c If a filter is applied, click the **X** next to *Filters Applied* to remove it and display all list items.

5 To save a search:

a Enter search criteria, then click **Apply**.

b Click **Save this search**.

c Provide a name and description for the search, then click **Save**. The search criteria is saved, and you can access it any time by selecting it in the *Saved Searches* drop-down list.

6 To access a saved search:

a On the left navigation bar, select the type of item the search relates to, then click the magnifying glass button in the upper right of the page.

b In the drop-down list at the top of the page, select a saved search. Only those searches related to the selected item type appear on the list.

Viewing policy version information

You can view the following version policy information:

- Creation date
- Modification date
- Deployment date

Procedure

1 Log in to CloudAz with an account that has permission to view policies.

2 On the left navigation bar, click **Policies.** The modification date for each policy is displayed.

3 Click the name of a policy.

4 On the policy detail page, in the left pane, click the version history icon. The policy creation date and deployment dates are displayed.

Modifying policies and components

As conditions change over time, you might want to change existing policies and components. For example, if a policy allows only Human Resources personnel to view offer letters, and the company CEO expresses an interest in viewing such letters, the policy would need to be modified to enable access for the CEO.

Procedure

1 Log in to CloudAz with an account that has permission to edit policies or components.

2 On the left navigation bar, click **Policies** or **Components > <Component type>.**

3 Click the name of the policy or component.

Note: If the name is grayed-out, you do not have sufficient privileges to modify the object.

4 On the policy or component detail page, make the desired changes, then click **Save or Save & Deploy**.

Next steps

If you chose *Save*, you must deploy the object before it can be used in policies or enforced. See Deploying components.

Changing user access to components

User access to components, policies, and other objects is determined by delegation policies. See Adding and editing delegation policies.

De-activating policies and components

You deactivate policies and components to un-deploy them.When you *deactivate* a a policy, it is removed from deployment and is no longer in active use. If you deactivate a top-level policy, all of its subpolicies are also deactivated.

You might want to deactivate:

- A component that was never used in any policy.
- A component that represents a concept that is no longer useful in your company; for example, a User Group component that represents the employees in a division that has been sold.
- A policy that is no longer useful. For example, a policy might cover documents that were confidential during project development, but have been made public since the project finished.

When policies are deactivated, they are removed from Policy Enforcement Points at the next heartbeat.

Procedure

1. Log in to CloudAz with an account that has permission to edit policies or components.
2. On the left navigation bar, click **Policies** or **Components.**
3. Locate the policy or component you want to deactivate, then select the deactivate button from the *Action* menu on the right side of the row. This deactivates the object. Policies that have been deactivated are removed from policy enforcers at the next heartbeat.

Deleting objects

After you deactivate a policy or component, it remains in the system so that you can reference it later. If you are sure you do not want to redeploy an item, you can delete it permanently.

Procedure

1. Log in to CloudAz with an account that has permission to view policies.
2. On the left navigation bar, click **Policies** or **Components**.
3. Locate the policy or component you want to deactivate, then select the trash can button.

1.5 Delegated administration policies

This section explains how to use delegated administration policies to manage administrator, designer, and user access to CloudAz.

Topics in this section

About delegated administration policies

Delegated administration policies enable administrators to control user access to the CloudAz user console by creating policies based on user attributes and conditions. This mirrors the way policies are used to control access to resources and eliminates the need to create and apply roles to users or groups of users.

For example, CloudAz users might have the following citizenship attributes:

- US
- Canada
- South Africa, California resident

An administrator could create the following delegation policies based on these attributes:

- US Citizens can manage policies tagged HIPPA or IPPA
- US Citizens can view policies tagged CMIA
- California residents can manage policies tagged CMIA
- Canadian Citizens can manage policies tagged NAPRA
- Canadian citizens cannot manage policies tagged HIPPA

When authorization access requests are received, the system evaluates the requests against the policies and responds accordingly.

Adding and editing delegation policies

Creating and editing delegation policies is similar to creating and editing the policies that govern access to resources. Administrators specify the required attributes, conditions, tags, and obliga-

tions for each delegation policy. Once created, delegation policies are used to determine CloudAz console access for users who have CloudAz user accounts.

Before you begin

- Decide what kind of policy you want to create, and whether to base the policy on user attributes such as locations or usernames.
- Know the attributes or usernames you want to use in the policy.
- Know the relevant tags applied to policy components.

Procedure

1 Log in to CloudAz with an account that has permission to manage delegation policies, such as the superuser Administrator account.

2 In the CloudAz console, select **Administration > Delegation Policies** on the left navigation bar.

3 Perform one of the following steps:

- To add a new delegation policy, click **ADD DELEGATION POLICY** in the upper right.
- To edit an existing delegation policy, click the name of an existing policy.

4 Add or edit the policy properties described in Table 1.5.1.

Table 1.5.1: Delegation policy properties

Option	Description
BASIC INFORMATION	The name and description of the policy. The name appears on the Delegation Policies list.
Name	The item display name. Multibyte characters, such as those used in Asian languages, are allowed, but names cannot exceed 128 characters and cannot include: ~ / * $ & \ ?
Description	A description of the item. This field is useful for explaining how the item is used and for documenting changes to the item. Descriptions can include multibyte characters, such as those used by Asian languages.
EFFECT & CONDITIONS	The policy actions and attributes that identify users.
Allow or Deny	The action taken when conditions in the policy are met. • **Allow**: Permit the listed Subjects to perform the task specified in the policy. Allowing a set of Subjects to perform a task does not mean that others are blocked from performing that action. Allow policies can never result in Deny responses. Allow policies can only result in Allow or Not Applicable responses. • **Deny**: Do not permit the listed Subjects to perform the task specified in the policy, but allow all others to do so. Deny policies can result in Deny, Allow, or Not Applicable responses.

Table 1.5.1: Delegation policy properties (Continued)

Option	Description
Conditions	User attributes used in the policy definition. User attributes include location, AD (Active Directory) Group, and Department. Select the appropriate operator (is, to include users that have the specified attribute, or is not to exclude users with the specified attribute). Then specify the attribute definition, such as USA for the Location attribute. Click **Add condition** to add an attribute.
MODULES	The CloudAz modules where users perform tasks. Use tags and conditional settings to further refine access to tasks. If no tags or conditions are specified, users are able to access all items. When you select an action check box, all required and optional action check boxes are automatically selected across modules. Table 1.5.2 lists the actions and the corresponding required and optional actions. You can review each module and choose to clear any action check box that is optional.
Policy	Controls access to policy-management tasks such as: • **Create Policy**: Set up new policies. • **Deploy Policy**: Make policies available to policy enforcers in the system. Use this action to deploy and deactivate policies. • **Delete Policy**: Remove policies from record. • **View Policy**: View policy details. • **Edit Policy**: Change the properties of existing policies.
Component	Controls access to policy-component-related tasks, such as: • **Create Component**: Set up new policy components. • **Deploy Component**: Make policy components available for use in policies. • **View Component**: View the properties of existing policy components. • **Delete Component**: Remove policy components from the system. • **Edit Component**: Change the properties of existing policy components.
Policy Model	Controls access to policy-model-related tasks, such as: • **Create Policy Model**: Set up new Policy Models. • **Delete Policy Model**: Remove Policy Models from the system. T • **View Policy Model**: View the properties of existing Policy Models. • **Edit Policy Model**: Change the properties of existing Policy Models.
Others	Controls access to administrative tasks, such as: • **Delegated Administration**: Manage delegated policies and users.Tasks related to managing policies that govern user access to CloudAz consoles. Users access these tasks by clicking **Administration > Delegation Policies** in the CloudAz left navigation bar. • **Reports**: Manage and view reports. Report-related tasks performed in the Reporter section of the CloudAz interface. Users with permission to create, delete, view, or edit reporter actions access these tasks by clicking **Reports** in the CloudAz left navigation bar. • **Tag Management**: Create Policy, Policy Model, and Component tags. • **Tools**: Test policies within the CloudAz console using Policy Tester.

Table 1.5.2: Required and optional actions

Action	Required actions	Optional actions
Create Policy	• View Policy • View Component • View Policy Model	• Edit Policy • Create Policy Tags
Delete Policy	• View Policy • View Component • View Policy Model	Not applicable
Deploy Policy	• View Policy • View Component • View Policy Model	• Deploy Component
Edit Policy	• View Policy • View Component • View Policy Model	• Create Policy Tags
View Policy	• View Component • View Policy Model	Not applicable
Create Component	• View Component • View Policy Model	• Edit Component • Create Component Tags
Delete Component	• View Component • View Policy Model	• Deploy Policy • Edit Policy
Deploy Component	• View Component • View Policy Model	Not applicable
Edit Component	• View Component • View Policy Model	• Create Component Tags
View Component	• View Policy Model	Not applicable
Create Policy Model	• View Policy Model	• Edit Policy Model • Create Policy Model Tags
Delete Policy Model	Not applicable	Not applicable
Edit Policy Model	• View Policy Model	• Create Policy Model Tags
View Policy Model	Not applicable	Not applicable
Manage Delegation Policies	• View Policy Model	Not applicable
Manage Users	Not applicable	Not applicable
Manage Reports	• View Reports	Not applicable
View Reports	Not applicable	Not applicable
Manage System Settings	Not applicable	Not applicable

Create Component Tags	• View Component
	• View Policy Model
	• At least one of the following actions:
	  • Create Component
	  • Edit Component
		Not applicable

Create Policy Model Tags	• View Policy Model
	• At least one of the following actions:
	  • Create Policy Model
	  • Edit Policy Model
		Not applicable

Table 1.5.2: Required and optional actions (Continued)

Action	Required actions	Optional actions

Create Policy Tags	• View Policy
	• View Component
	• View Policy Model
	• At least one of the following actions:
	  • Create Policy
	  • Edit Policy
		Not applicable

Policy Tester	• View Policy • View Component • View Policy Model	Not applicable

5 Click **Save** in the upper right. The policy is used to determine console access for users who have CloudAz user accounts.

Next steps

Add accounts for the users to be governed by the policy. See Managing user accounts on page 145.

Deleting delegation policies

Deleting delegation policies removes them from the CloudAz system.

Procedure

1 Log in to CloudAz with an account that has permission to manage delegation policies, such as the superuser Administrator account.

2 On the left navigation bar, select **Administration > Delegation Policies**.

3 Select the policies you want to delete, then click the trash can button.

Managing user accounts

User accounts enable users to log in to the CloudAz console using a specified username and password.

Important: Delegation policies determine the tasks users can perform when they are logged in to CloudAz. If delegation policies prohibit users from performing any tasks in the CloudAz interface, users can log in using CloudAz user accounts, but their activities are limited to viewing the static dashboard and the *Getting Started* page. See Adding and editing delegation policies.

Adding or editing user accounts

Administrators with permission to manage Administrator Actions, including the Create Administrator and Edit Administrator tasks, can add or edit CloudAz user accounts as needed. These accounts enable users to log in to the CloudAz console.

Procedure

1 Log in to CloudAz with an account that has permission to Create and Edit Administrators in the *Administration* section of the CloudAz console.

2 On the left navigation bar, click **Administration > Users**.

3 Do one of the following:

- Click **Add User**.
- Click the name of an existing user.

4 Add or edit the user account properties:

Table 1.5.3: User account properties

Option	Description
User Information	The user's first and last name and username. The first and last name appear on the *Users* list.
First Name	The first name of the user.
Last Name	The surname of the user.
Username	The name the user enters to log in to CloudAz. You cannot change the username after the account has been added.
Email	The user's email address.
Password and Confirm Password	The password the user enters to log in to CloudAz.Passwords must be between 7 and 12 characters and must contain at least one number, one character, and one non-alphanumeric character other than '_'. **Note**: You cannot change the password of user accounts after they have been created in the Administration > Users section of the console. In addition, users must change their passwords when they log in to the console for the first time. The password policy prevents users from reusing the last 5 previously used passwords.
User Attributes	User characteristics that can be referenced in delegation policies.

5 Click **Save**. The account is added or updated, and users can log in with the specified credentials immediately. If the username of an existing account has been modified, users who are logged in can continue their session, but they must provide the updated username at the next login.

The account is added or updated.

Deleting user accounts

Administrators with permission to manage Administrator Actions, including the Delete Administrator task, can delete CloudAz user accounts as needed.

Procedure

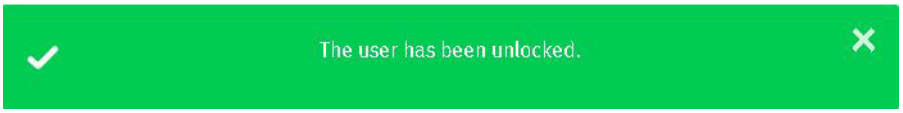
1. Log in to CloudAz with an account that has permission to delete user accounts in the *Administration* section of the CloudAz console.
2. On the left navigation bar, click **Administration > Users**.
3. Select the user account to be deleted, then click the trash can button. The user account is removed from the CloudAz system.

Unlocking a user account

To prevent the try-and-error password guessing, a user is locked out of their account after 5 unsuccessful login attempts. When the user is locked out, the user cannot log in even with correct credentials. Even if the user gets the password correct on the sixth try, the account will still be locked. In the event of a user lockout, the user needs to request that the system administrator unlock the account.

Procedure

1. Log in to CloudAz with an account that has permission to manage users, such as the superuser Administrator account.
2. On the left navigation bar, click **Administration > Users > <*name of user*>**.
3. In the row corresponding to the user's name, click the padlock button  to unlock the user account. An unlock confirmation message appears.



4. Inform the user that the account has been unlocked.

Changing the superuser password

The superuser *Administrator* account cannot be deleted or renamed. However, anyone logged in to the CloudAz console with the superuser account can change the superuser password as described in this section.

Procedure

1. Log in to CloudAz as the superuser Administrator.
2. In the drop-down list in the upper right of the screen, select **Change Password**.

3 Provide the old and new passwords, then click **Submit**.

The password for the console and all utilities is changed.

Part 2 Managing Reports

2.1 Introducing Reporter

This section describes Reporter, the console used to monitor and report on policy compliance, gather statistics about document usage, and investigate any suspected incidents of information mishandling. Reporter is used by administrators, IT staff, managers, executives, auditors, and other authorized personnel.

Topics in this section

About Reporter

Reporter is a web-based interface that is installed as a component of the console. You use Reporter to define and run custom queries about policy enforcement activities that are recorded in the Activity Journal. These queries are referred to as *reports*. Reports can be designed to answer a wide variety of questions, such as who has access to certain documents; who is using which resources, and when; what types of policy enforcement is taking place; what activity occurred within a given department; and so on.

In addition to reports, you can use Reporter to create monitors that trigger alerts when specified policy enforcement criteria are met. You can design monitors to cover a wide range of scenarios, such as sending an alert through email when access to a specified resource has been denied more than a specified number of times in a given time period; or when the classified documents that have been downloaded in a given time period exceed a specific file size. Together, monitors and alerts provide continuous coverage of critical policy enforcements in an enterprise, as well as a notification system that alerts administrators when action is required.

Accessing the Reporter console

Access the Reporter console from the CloudAz console. Access to the Reporter console is determined by delegated administration policies. See Adding and editing delegation policies.

Procedure

1 Open a web browser and log in to CloudAz.

Note: If you are using the Firefox browser, you enable must enable TLS 1.0 (under **Tools > Options > Advanced**) to access Reporter.

2 On the left navigation bar, click **Reports**. The Reporter console opens in a new browser window.

About the Reporter console

The Reporter console has three tabs: **Dashboard**, **Reports**, and **Monitoring**. By default, the *Dashboard* tab appears when Reporter opens. Only users granted access to monitors can access the *Monitoring* tab.

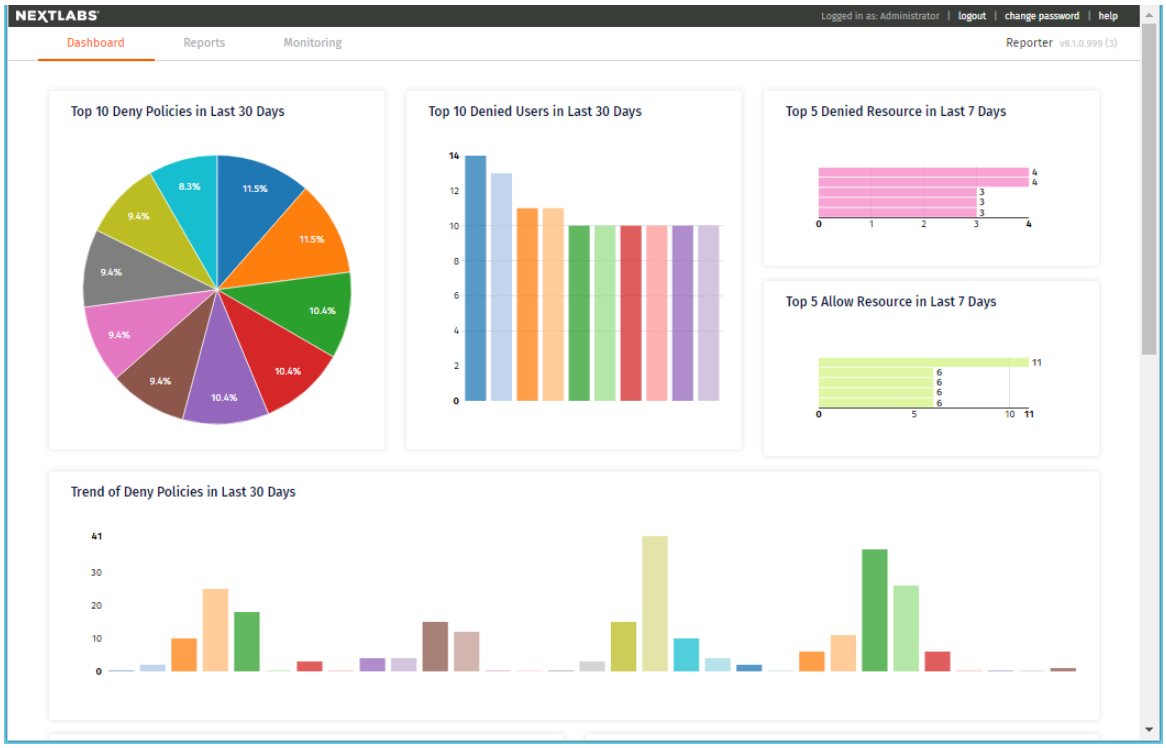


Figure 2.1-1: Reporter console

Dashboard tab

The *Dashboard* tab displays charts and tables representing the results of a set of predefined queries. Data is refreshed from the Activity Journal each time you open the Dashboard. This means that data is updated on demand. Also, if a section shows a statistic for the past week, that reflects not the last seven whole calendar days, but the last seven 24-hour periods starting from the top of the current hour.

Reports tab

The *Reports* tab is where you define and run reports, save reports for future use and, optionally, share reports with other users. See Working with reports.

Monitoring tab

The *Monitoring* tab is where you define monitors to receive notifications (alerts) about specific policy enforcements and view alerts triggered by the monitors. Only users granted access to monitors can access the *Monitoring* tab. See Working with monitors and alerts.

Banner

The banner at the top of the web page displays version information and links.

Logout

The **logout** link enables you to log out of the Reporter console. You can also log out by closing your web browser, or closing the active tab in your web browser.

Help

The help link provides information to help you use Reporter.

2.2 Using the Reporter Dashboard

This section describes the Reporter Dashboard. Reporter is the console used to monitor and report on policy compliance, gather statistics about document usage, and investigate any suspected incidents of information mishandling. Reporter is used by administrators, IT staff, managers, executives, auditors, and other authorized personnel.

Topics in this section

Introducing the Reporter Dashboard

The Reporter Dashboard is divided into panes, each displaying a predefined statistical view of data that provide a snapshot of policy enforcement trends. In the default configuration of Reporter, the following information is displayed:

- Most commonly Denied requests for access to data resources, by resource, user, and policy
- Most commonly Allowed request for access to resources, by resource
- 30-day trend for the enforcement of all Deny policies, for all users and resources
- Detailed information about the most recent instances of any policy either Allowing or Denying a user access to any resource
- Statistics on alerts in the current week
- Detailed information about alerts raised in the current day

What you see may be different if the application administrator changed the data that appears in the Dashboard or its layout.

The data displayed in all panes of the dashboard is refreshed from the Activity Journal each time you open the *Dashboard* tab. This means that data is updated on demand. Also, if a section shows some statistic for the past week, that reflects not the last seven whole calendar days, but the last seven 24-hour periods starting from the top of the current hour.

Figure 2.2-1 shows a sample dashboard. In the CloudAz console, the charts have interactive features. When you place the mouse pointer over a bar in a bar chart or over a slice in the pie chart, a tooltip displays information about that value series.

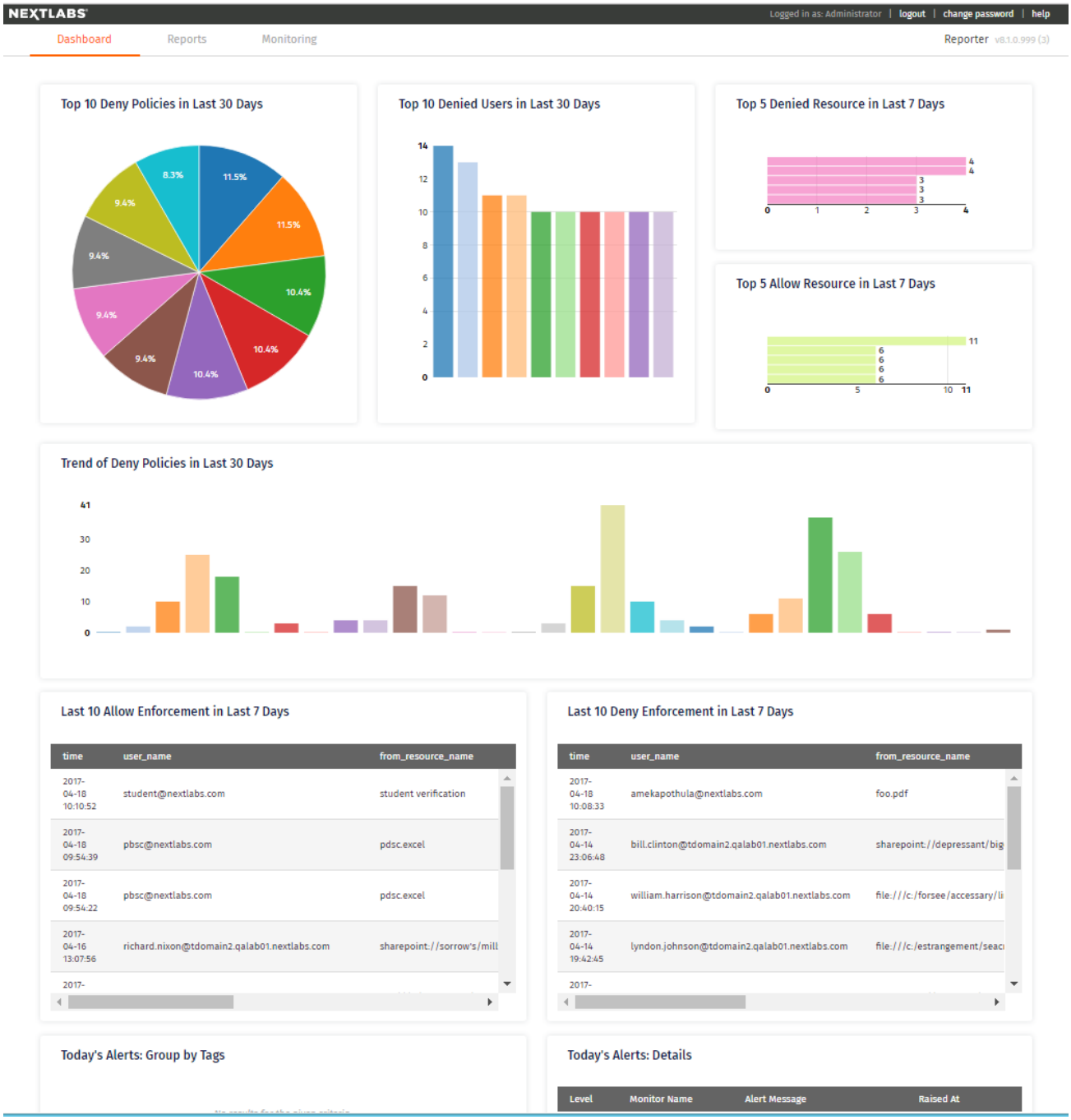


Figure 2.2-1: The Dashboard tab

About Reporter Dashboard data

Table 2.2.1 summarizes the data presented in each of the predefined queries represented on the Dashboard, and discusses the possible implications of and uses of the data displayed in each.

Table 2.2.1: Reporter Dashboard data

Information/ section	Description	May Indicate:
Top Deny Policies (30 days)	Pie chart representing the Deny policies that were most frequently enforced over the previous thirty days.	• Misunderstanding of access level: users being blocked from a resource they believe they should use • Incorrectly defined entitlements: users should have access, but policies are not updated or correctly designed
Top Denied Users (30 days)	Bar chart representing the users who have had the most instances of any Deny policy enforced against them.	• Users who habitually snoop into resources they are not authorized to use • Incorrectly defined entitlements: users or group should have access, but policies are not updated or are incorrectly designed
Top Denied Resources (7 days)	Bar chart representing the resources that any users have most frequently attempted to access and been blocked by an active policy, over the previous seven days.	• Resources of broad interest to users who should not be using them • Incorrectly designed resource or user component, blocking users who should have access
Top Allow Resources (7 days)	Bar chart representing the resources that users have most frequently attempted to access and been allowed by an active policy, over the previous seven days.	• Improperly designed resource component or policies, which allow inappropriate users access to sensitive resources
Deny Policy Enforcement Trends (30 days)	Bar chart representing the trend, over the previous 30 days, of the daily total instances of any deny policy being enforced on any user, for any resource.	• Progress (or lack of it) in educating users about access policies and individual/group entitlements, at a broad level • Improperly designed policies that are blocking too many users who expect and are entitled to access or use
Last 10 Allow Enforcement in Last 7 Days	List of details about the most recent instances of any allow policy being enforced against any user, for any resource. Details listed include: • time: Date and time of enforcement • user_name: The name of the user who triggered the policy • Resource Id: The resource the user was targeting • policy_decision: The authorization decision • action: The action that triggered the policy	• Instances where some urgent action is required, such as users being allowed access to some resource they should not be using, due to lack of policy coverage or an incorrectly defined policy
Last 10 Deny Enforcement in Last 7 Days	List of details about the most recent instances of any deny policy being enforced against any user, for any resource. Details listed include: • time: Date and time of enforcement • user_name: The name of the user who triggered the policy • Resource Id: The resource the user was targeting • policy_decision: The authorization decision • action: The action that triggered the policy	• Instances where many users are attempting to get at data they are not authorized to use • Instances where some urgent correction is required to allow appropriate access, such as multiple authorized users being blocked from some resource they need by an incorrectly defined policy

Table 2.2.1: Reporter Dashboard data (Continued)

Information/ section	Description	May Indicate:
Today's Alerts: Group by Tags	Treemap representing volume of alerts. Alerts are grouped by monitor tags.	• Policies being watched by monitors that are tagged, are being enforced at a rate that demands attention. Further review or action may be required.
Today's Alerts: Details	List of details about the alerts raised in the current day. Details include: • Level • Monitor name • Alert message • Date and time the alert was raised	• Policies being monitored are being enforced at a rate that demands attention. Further review or action may be required.

2.3 Working with reports

This section describes how to use Reporter to define, save, and run reports about data, resource access and use, and policy enforcement throughout your enterprise. Reporter is the console used to monitor and report on policy compliance, gather statistics about document usage, and investigate any suspected incidents of information mishandling. Reporter is used by administrators, IT staff, managers, executives, auditors, and other authorized personnel.

Topics in this section

Reporting functionality and permissions

You define and run reports in the *Reports* console. The functionality available to you depends on the rules established in the *Delegated Administration* section of the console. See Delegated administration policies.

The *Reports* tab has two panes. The pane on the left displays the **Saved Reports**—the list of all saved reports available to you. This includes all reports you create and save (if you have permission to create reports), all reports saved by other users and shared with you, and the sample reports used to generate data that is displayed in the *Dashboard* tab.

When you click on any item in **Saved Reports**, the details of that report display in **Report Details**, on the right. This is also where you work when you create a report. In **Report Details**, you define the following:

- The time period of the policy activity data to cover in the report
- The criteria, or filters, that identify the policy activity data to include in the report
- The output format of the report

The default settings in **Report Details** appear when you click the *Reports* tab or when you click **New** in the **Saved Reports** pane. By default, the time period of the report is the current day, all policy activity data at the user level is included, and the data is presented in table format.

After defining a new report or editing an existing report, click **Run** at the bottom of the **Report Details** pane to view results, as shown in Figure 2.3-1.

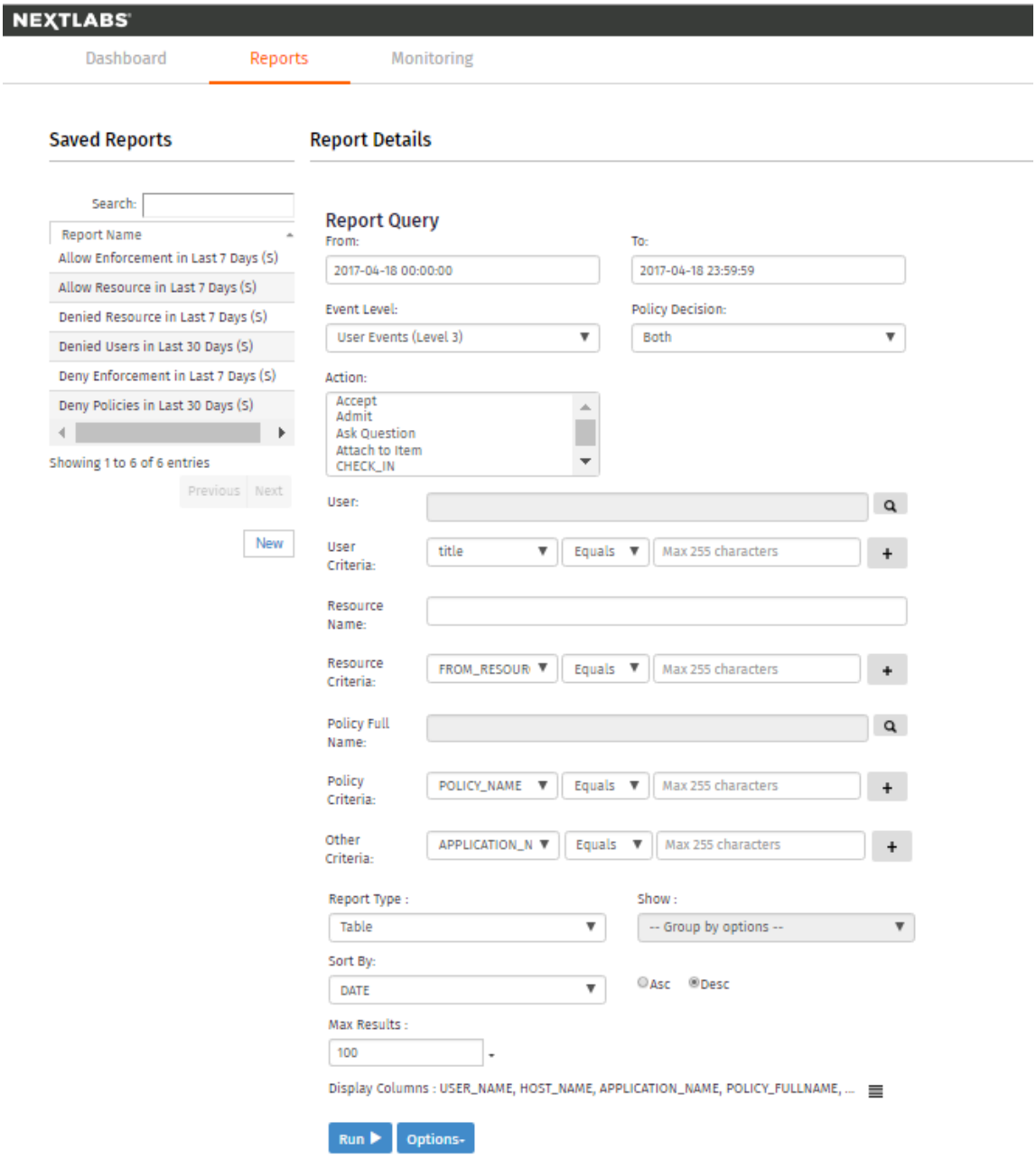


Figure 2.3-1: The Reports tab

Figure 2.3-2 shows an example of a generated report. In the example, the output format is a bar chart. The chart appears at the bottom of the **Report Details** pane. Depending on the output format you select in **Report Type** and the browser you are using, the generated report appears either in the **Report Details** pane or in a separate browser tab.

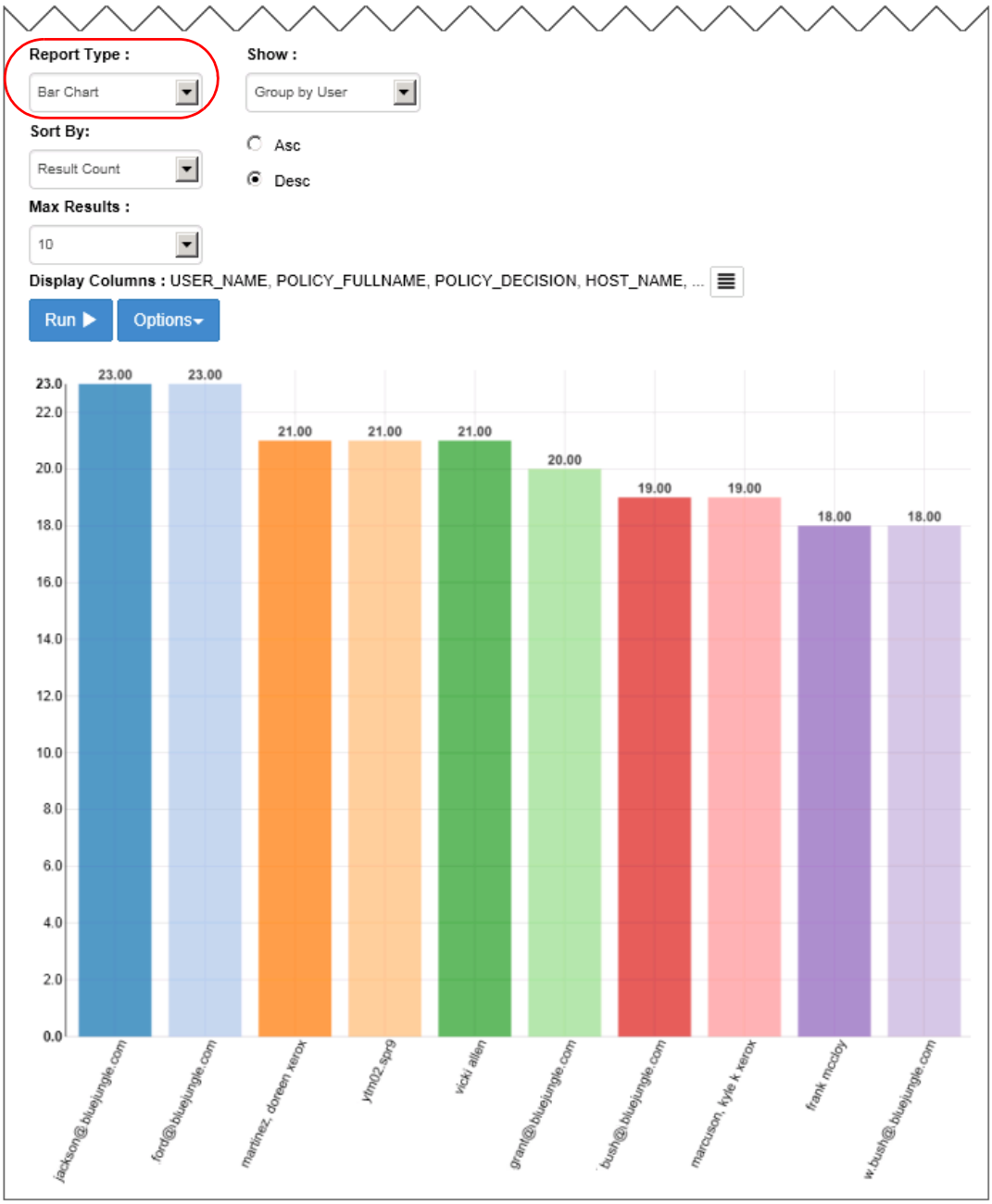


Figure 2.3-2: Report Results (bar chart) at the bottom of Report Details

Creating a report

A report is a formatted view of a particular set of information extracted from the Activity Journal, a set of database tables where policy activity data is logged. You create a report by defining a query—criteria, or filters, for the policy activity data you want to include in the report—and specifying the display options that determine how the data is presented.

Just as policies can be designed to control information access and usage by types of users, actions, resources, applications, and computers, you can design report queries that filter policy activity data by attributes of those components. The greater the number of filters, the fewer the number of records returned. For example, compare the following queries:

- Retrieve data for deny enforcements of any policy (in other words, all policies), covering any action by any user on any resource, for one week.
- Retrieve data for deny enforcements by communication policies that cover only the email action on only documents classified as Confidential, for one week.

The first query specifies fewer conditions and retrieves more data than the second.

Important: If you don't specify any filters or change any of the default settings, the report retrieves all policy activity data categorized as user-level events, for the current day.

Procedure

1 In the CloudAz console, click **Reports** on the left navigation bar.

2 Click the **Reports** tab, if it is not already open.

3 If the query of an existing report is currently displayed in the **Report Details** pane, click **New** at the bottom of the **Saved Reports** pane.

4 Define the report query and display options. Many of the fields are optional. Required fields are populated with default values.

Table 2.3.1: Report query and display fields

Query field	Description
From and To	The start date and time, and end date and time, respectively, of the time period the report covers. Click in the field to choose a date and time from the calendar. When specifying a report period, consider the time zone of the system and the time period of data stored in the Activity Journal. See Specifying a report period.
Event Level	The level of event verbosity the report contains: • User Events (default) • Application Events (application and user-level events) • All System Events (system, application, and user-level events) As a rule, you should leave this setting at **User Events**. This setting significantly reduces the amount of system noise. Application- or system-level events are generally not useful in monitoring policy or user activities.

Table 2.3.1: Report query and display fields (Continued)

Query field	Description
Policy Decision	The type of enforcement effect to include in the report: • **Allow**: Instances when the policy permitted the user to perform the action covered by the policy. Report results are determined by the information that is logged. For example, if the policy contains no On Allow logging obligations, the report returns no On Allow data regardless of whether you select this option. • **Deny**: Instances when the policy did not allow the user to perform the action. Deny decisions are always logged. • **Both:** All instances when the policy was enforced, with either Allow or Deny effect.
Resource Type	The resource type to include in the report. Resource types are the templates that include information about attributes, actions, and obligations available to components and policies.
Action	The user action or actions to include in the report. The list shows all currently defined actions for the resource type. To select multiple actions, press **Ctrl** and click each action. If no actions are selected, all actions are included in the report. Policies involving Paste actions do not support logging obligations, therefore, instances of their enforcement are not included in reports.
User	One or more users on which to filter the activity data, or leave this field blank to include all users. To browse for users in your directory, click the magnifying glass icon to use the **User Lookup** window.
User Criteria	Conditions used to specify additional user criteria. Each condition consists of a user attribute, an operator, and a value. You must click the + button to add a condition to the query. See Filtering by criteria.
Resource Criteria	Conditions used to specify additional resource criteria. Each condition consists of a resource attribute, an operator, and a value. You must click the + button to add a condition to the query. See Filtering by criteria.
Policy Full Name	One or more policies on which to filter, or leave this field blank to include all policies. Use the Policy Lookup window to browse through and select the policies.
Policy Criteria	Conditions used to specify additional policy criteria. Each condition consists of a policy attribute, an operator, and a value. You must click the + button to add a condition to the query. See Filtering by criteria.
Other Criteria	Conditions used to specify additional criteria. Each condition consists of a general attribute (for example, hostname, host IP, and application name), an operator, and a value. You must click the + button to add a condition to the query.See Filtering by criteria.
Report Type	The output format in which to display the data: Table, Bar Chart, Horizontal Bar Chart, or Pie Chart. Use a table to display policy activity details in a row-and-column format. Use a chart to display a summary of policy activities. For examples of reports in different formats, see Sample reports. For chart report types, select grouping options (grouping is not available to table type reports): • Group by User: The chart shows the number of enforcement events for each user covered by the report. • Group by Resource: The chart shows the number of enforcement events for each resource covered by the report. • Group by Policy: The chart shows the number of enforcement events for each policy covered by the report. • Group by Month: The chart shows the number of enforcement events for each month covered by the report. Select this option only if the time period you specified spans more than one month. • Group by Day: The chart shows the number of enforcement events for each day covered by the report.
Show	The display options for the report type (not available for Table type reports).

Table 2.3.1: Report query and display fields (Continued)

Query field	Description
Sort By	The field on which to sort the data, and select **Asc** to sort in ascending order or **Desc** to sort in descending order. If the report type is a table, you can sort the data by any attribute. If the report type is a chart, you can sort either by the grouping item (user, resource, policy, month, or day) or by Result Count (the number of enforcement events for each user, resource, policy, month, or day).
Max Results	The maximum number of results to display in the table or chart. For charts, this number represents the maximum number of bars in a bar chart, or slices in a pie chart. For readability reasons, charts should display a limited number of bars or slices. For a table, the number represents the maximum number of rows (each row represents an event). Tables that show a large number of rows present the data on multiple pages.
Display Columns	The columns to display in a table. This setting applies to a table only. USER_NAME, POLICY_FULLNAME, POLICY_DECISION, HOST_NAME, and APPLICATION_NAME are selected by default. To remove any of those columns or to add other columns, click  and use the arrow icons to move columns out of, or into, the Selected pane.

5 Click **Run** to generate the report.

6 If you want to run report at a future time, save it by clicking **Options** > **Save**. See Saving reports.

Specifying a report period

When specifying the report start date and time and the end date and time, be sure to consider the time zone of the system and the length of time that data is stored in the Activity Journal.

Time zone considerations

The report period is determined using the time zone of the CloudAz system, not the time zone where you are running the Reporter console or the time zone where the events you are querying on occurred. CloudAz uses the time zone of the region specified for the subscription. Trial subscriptions use Pacific time.

To generate reports for specified periods, you might need to add hours to the beginning or end of your time period, depending on the time difference between the CloudAz time zone and the location where the events of interest occurred.

For example, if your CloudAz system uses the Eastern time zone, and you want to query on events in the Pacific time zone, be aware that there is a three hour difference between the time-zones. If you specify date-time values that start and end at midnight, your query returns three hours of data from the night before the first date in your time period (Pacific time), and does not include the last three hours of data on the final day of the time period.

Activity Journal archive considerations

By default, the Activity Journal stores data for 90 days before archiving it, but this setting can be changed by the database administrator. Depending on the archive setting, data for the period you specify might be unavailable. If a query uses all default values, except for the report period, and it returns no data, try changing the **From** and **To** values to later dates.

To create a report that uses data that is already archived, ask your database administrator to restore the data to the Activity Journal.

Filtering by user, resource, and policy

When filtering policy activity data by user, resource, or policy, you can use either, or both, of the following methods:

- Filter by one or more usernames, resource paths, or policy names
- Filter by user, resource, or policy criteria

Filtering by name

To filter by one or more usernames, resource paths, or policy names, you specify values in the **User**, **Resource Path**, and **Policy Name** fields.

For **User** and **Policy Name**, use the Lookup tool to browse for and select the names. Figure 2.3-3 shows the User Lookup window. The Policy Lookup tool is similar.

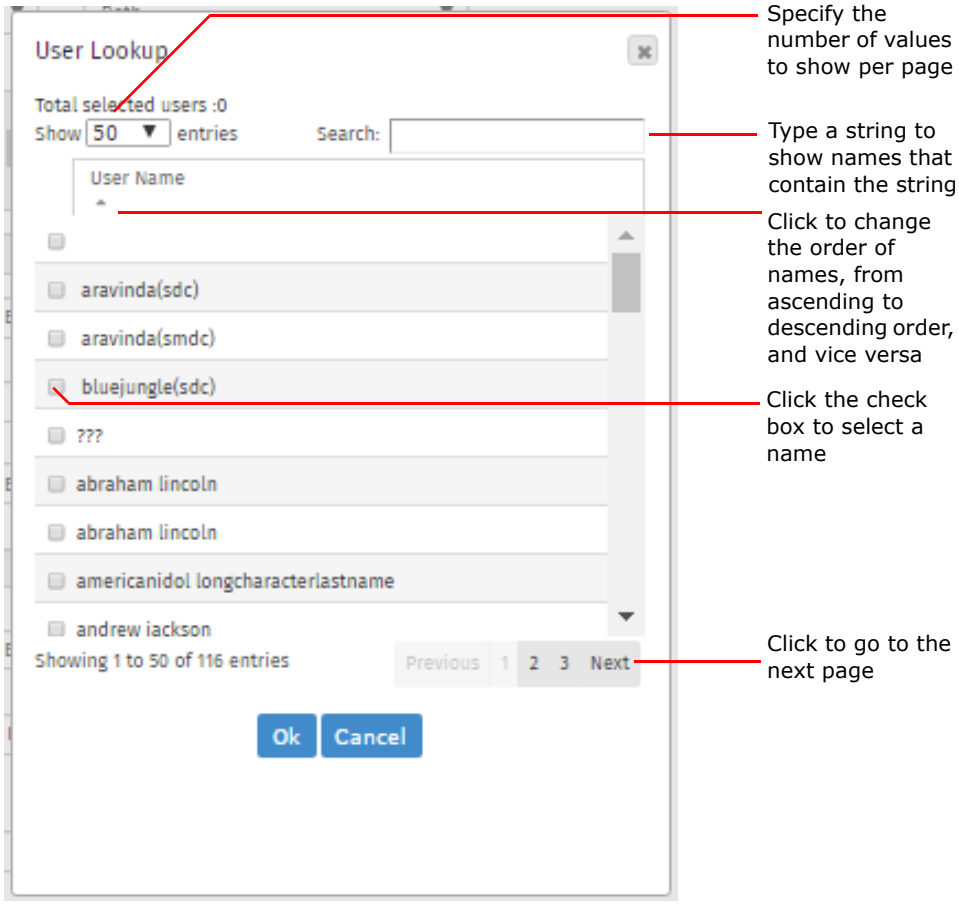


Figure 2.3-3: User Lookup window

There is no lookup tool for **Resource Path**. You must type the resource path value. Make sure you enter the correct information. If you specify a path that does not exist, the filter essentially has no effect. Observe the following guidelines when typing values:

- For multiple values, separate each value with a comma.
- Include the full folder path in each value.
- Use upper- or lowercase characters as needed. Queries are not case sensitive.
- Use an asterisk (*) as a wildcard character. For example, you can type the following string to specify all content—subfolders and files—in the `Finance` folder:

  ```
  c:\Finance\*
  ```

Filtering by criteria

To filter by user, resource, or policy criteria, use conditions in the **User Criteria**, **Resource Criteria**, or **Policy Criteria** fields. Each condition consists of an attribute, an operator, and a value. You can create multiple conditions for each filter type. For example, you can filter resources by the Modified By and Keywords attributes to find documents modified by a particular author *and* that contain a specified keyword. Both conditions must be true for the filter to return any results.

You can create a condition for **User Criteria**, **Resource Criteria**, or **Policy Criteria**.

Procedure

1. In the CloudAz console, click **Reports** on the left navigation bar.
2. Click the **Reports** tab, if it is not already open.
3. On the Report Query form, select an attribute from the Resource Criteria drop-down list. The list displays the attributes that have been defined. The policy attributes list also includes the tags, if any, defined for policies.

   Figure 2.3-4 shows an example of a list of resource attributes.

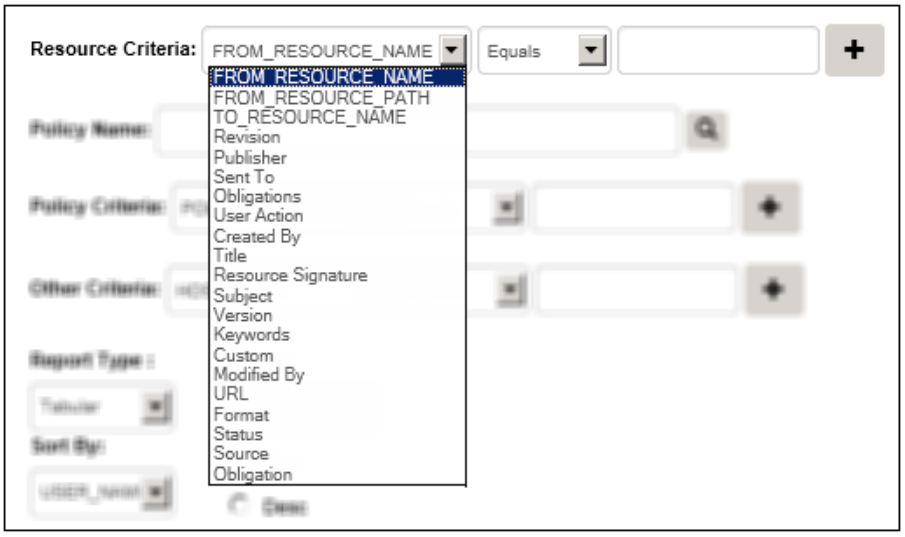


Figure 2.3-4: Resource attributes list

4 Select a comparison operator from the second drop-down list.

- **Equals**: Attribute value matches a specified value.
- **Not Equals**: Attribute value does not match a specified value. Use this operator to exclude the specified value from the result set.
- **Like**: Attribute value is similar to the specified characters. This operator is similar to the SQL LIKE keyword. Use it to match a specific pattern, or when you do not know the entire value.
- **In**: Attribute value matches any one of a specified set of values. This operator is similar to the SQL IN keyword. Use it to create an OR condition, that is, when you want to match multiple values.

For all the operators, searches are case-insensitive. For example, *Patrick Marleau* is the same as *patrick marleau*.

5 Type a value in the third field.

- If you selected the Equals operator, the specified value must be an exact match (aside from case); otherwise, the filter does not find any matches and the report does not return any results.
- If you selected the Not Equals operator, the specified value you do *not* want to match must also be an exact match to an attribute value (aside from case); otherwise, the filter does not exclude any values, and you get more results than you expect.
- If you selected the Like operator, you can use the following wildcard characters:
  - %: matches one or more characters.
  - _: matches any single character. You can use more than one underscore consecutively; for example, to match any two consecutive characters, use two underscores.
- If you selected the In operator to specify a set of values, separate each value with a comma.

For examples of conditions that use these different operators, see Filter examples.

6 Click the + button to add the condition to the report query.

7 Repeat the previous steps to add conditions as needed. Figure 2.3-5 shows an example of a query where two conditions have been created for **Resource Criteria**. To remove a condition, click the [—] button.

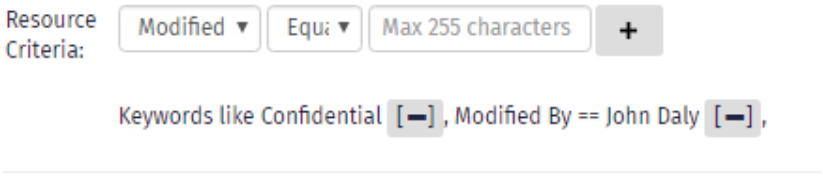


Figure 2.3-5: Two conditions in Resource Criteria

Filter examples

The following condition searches for resources that contain the keyword Confidential (the search is not case sensitive):

```
Keywords Equals Confidential
```

The following condition searches for users who are not US citizens:

```
ISO Country Code Not Equals US
```

The following condition searches for users whose email addresses end with acme.com:

```
Email Address Like %acme.com
```

The following condition searches for machine names that start with NL, followed by any two characters (represented by two underscores), the letters CC, and other characters, for example, NL03CC-Win64:

```
HOST_NAME Like NL__CC%
```

The following condition searches for users who are in the Accounting or Finance departments:

```
Department In Accounting, Finance
```

Filtering by action

You can filter the policy activity data by user actions, such as create, run, or rename. The Action list displays all the actions that have been defined. You can select more than one action from the list, for example, for a report on all instances of users moving, renaming, or deleting certain files and those actions triggering policy enforcement.

Running a report

You can run any report selected from **Saved Reports**, or a new report you define in **Report Details**. Simply click the **Run** button at the bottom of **Report Details**.

When you run a report, Reporter compares the information currently in the Activity Journal to the report criteria, selects the matching information (if any), calculates the totals, and displays the results in the format you selected.

If you selected a chart as the output format, the chart appears at the bottom of the **Report Details** pane. Scroll down to view the chart. If the output format is a table, it might appear in a separate tab in the browser window, depending on the browser you are using. Figure 2.3-6 shows an example of policy activity data displayed in a table.

Showing page 1 of 7 Go to page:

Date	USER_NAME	POLICY_FULLNAME	POLICY_DECISION	HOST_NAME	APPLICATION_NAME
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\userinit.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\ServerManagerLauncher.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\dwm.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\explorer.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 5:51 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\wlrmdr.exe
Aug 7, 2014 6:04 PM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\wermgr.exe
Aug 8, 2014 1:51 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 1:52 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 1:53 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 1:54 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 1:55 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 1:56 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 8, 2014 9:51 AM	Administrator@DOC-W2K8CC	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\taskhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Program Files\NextLabs\Desktop Enforcer\bin\nlsce.exe
Aug 7, 2014 1:28 PM	Administrator@TDOMAIN2	/test folder/new policy	Denied	doc-w2k8cc.tdomain2.nextlabs.com	C:\Windows\System32\conhost.exe

Figure 2.3-6: Report results in table format

If no data matches the specified criteria, a message appears, as shown in Figure 2.3-7.

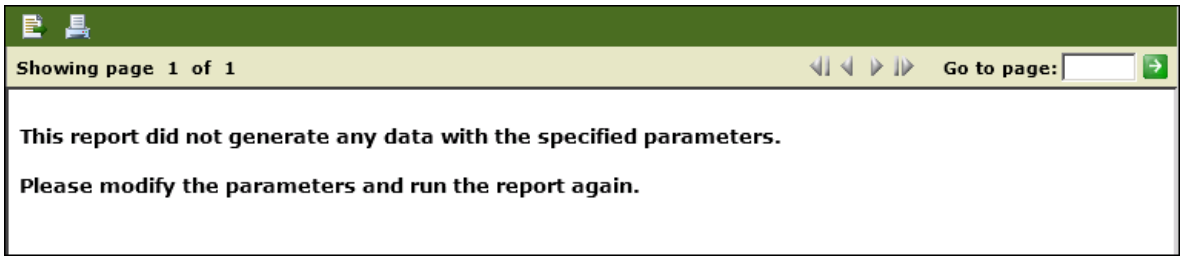


Figure 2.3-7: No data message

Return to the report query to check the specified criteria, and modify the query as needed. Check the time period. As discussed in Activity Journal archive considerations, if the specified time period is too far in the past, the data is likely to have been archived. Remember, too—the greater the number of filters, the narrower the scope of data returned. If there are conditions specified in multiple fields, such as **User Criteria**, **Policy Criteria**, and **Resource Criteria**, all the conditions must be true for the data to match.

Conversely, if the report returns more data than you expect, check the filters in the query. For example, verify that the filters have been added to the query. Figure 2.3-8 shows an example of a resource filter that has been defined (values are entered in the **Resource Criteria** fields), but that has not been added to the query. You must click the + button to add a filter. When a filter has been added, it appears below the field, as indicated by the policy filter that appears below **Policy Criteria**.

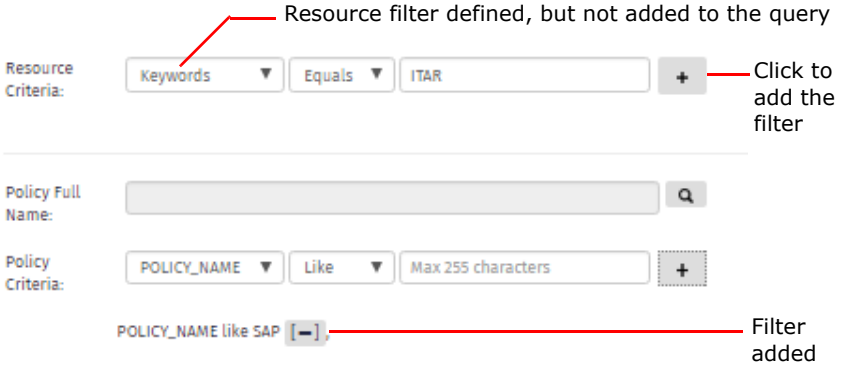


Figure 2.3-8: Filter examples, one added and one not

Viewing report output

When viewing the report output, you can drill down for additional information. A table shows all the policy enforcement events that meet the criteria specified in the report query. You can click the date value in any row to get details for an event. The details appear in a Log Details Report on a separate page, as shown in Figure 2.3-9.

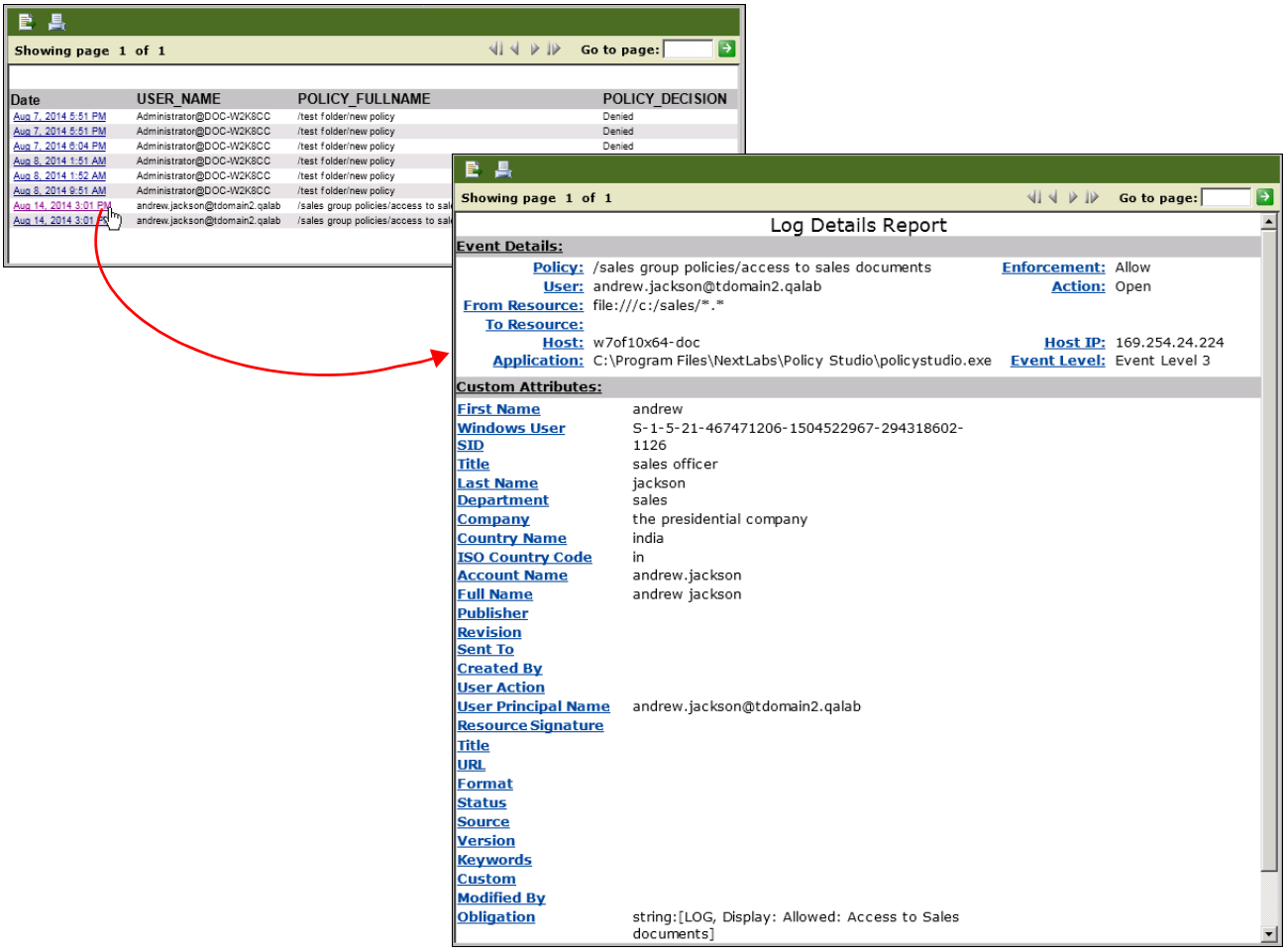


Figure 2.3-9: Selecting a row to obtain event details

Unlike a table, a chart does not show each policy enforcement event. Rather, a chart aggregates policy enforcement events by the grouping option you specify. For example, a chart can group events by user. In this case, each slice of a pie chart or bar in a bar chart represents a user and the total number of events triggered by the user. You can click a chart slice or bar to see all the events triggered by a particular user.

Figure 2.3-10 shows the list of events that appear when a bar in a bar chart is clicked. The number, 5.00, on top of the bar indicates the number of events for a particular user, and, correspondingly, the table lists the five events. Drill down for details about each event by clicking a date value in a row in the table.

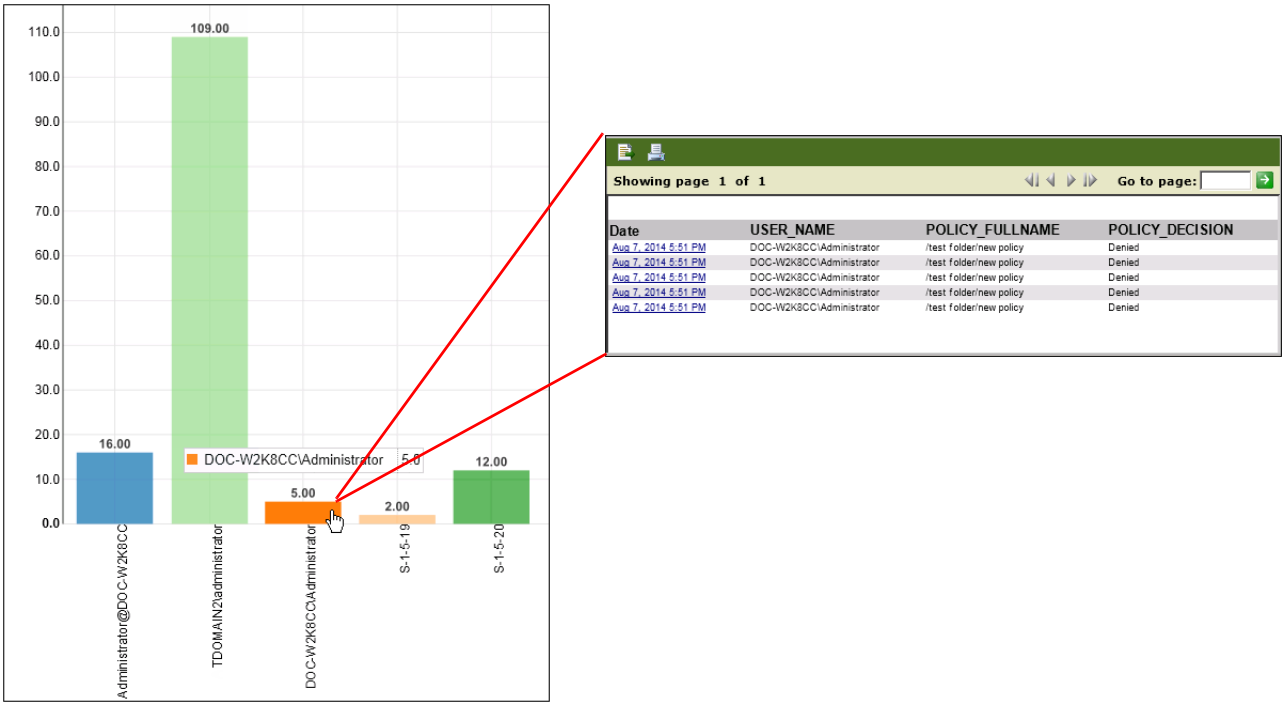


Figure 2.3-10: Clicking a bar in a chart to obtain a list of events

Non-Latin characters

If you are using Microsoft SQL and your database includes columns with content in languages that use character sets other than standard Latin, reports based on them might not display properly. For example, they might show question marks instead of the proper characters.

If you encounter this problem in reports, you can solve it by having your database administrator change the collation value for the columns in question, to match the language of the content of that column. Note that only non case-specific collations should be used, whatever the character set. Once you do this, the reports should generate properly.

This should be done on a column-by-column basis, and applies only to Microsoft SQL databases.

Saving reports

When you create a report, it is not saved automatically. Similarly, if you change any settings in a report you selected from **Saved Reports**, the changes are not saved until you explicitly save the report. This behavior is convenient for creating and running reports on an ad hoc basis. However, to run a report regularly—for example, for a weekly status report on policy activity related to certain export-controlled documents stored on specific servers—you can save the report.

Another reason to save a report is to share it with other users. The actions that users can take with a shared report depend on their permissions. Administrators can run the report with different criteria, save a new version of the report and share it with others. Non-administrative users can only run and view the report. They cannot modify the report criteria, but they can select a different report format, for example, a different chart type.

You can save a new and modified reports as needed.

Procedure

1. At the bottom of the **Report Details** pane, click the **Options** button, then select **Save**. If you are saving changes made to a previously-saved report, you can select **Save As** to save the report as a new report. The **Save As** command is your only option if you are saving a report that another user created.
2. In the **Save** dialog, specify the following information:
   - In **Name**, type a name for the report.
   - In **Description**, provide a clear description of the report, particularly if you plan to share the report with other users.
   - In **Date mode**, select one of the following values:
     - **Fixed Dates**: The report uses the dates currently specified in the **From** and **To** fields.
     - **Relative Dates**: The report uses dates relative to the date the report is run. This is the typical option for running a report on a regular schedule, for example, weekly, monthly, or quarterly.
3. If you selected **Relative Dates**, select one of the following values in **Date Selection**:

Value	Description
Today	The report searches for data for the current day.
Current Week	The report searches for data in the week in which the report is run. For example, if the report is run on 2014-09-15, it searches for data from 2014-09-14 (Sunday) to 2014-09-20 (Saturday).
Current Month	The report searches for data in the month in which the report is run. For example, if the report is run on 2014-09-15, it searches for data from 2014-09-01 to 2014-09-30.
Current Quarter	The report searches for data in the quarter in which the report is run. For example, if the report is run on 2014-09-15, it searches for data from 2014-07-01 to 2014-09-30.
Yesterday	The report searches for data from the previous day.
Last Calendar Week	The report searches for data in the previous week. For example, if the report is run on 2014-09-15, it searches for data from 2014-09-07 to 2014-09-13.
Last Calendar Month	The report searches for data in the previous month. For example, if the report is run on 2014-09-15, it searches for data from 2014-08-01 to 2014-08-31.
Last Calendar Quarter	The report searches for data in the previous quarter. For example, if the report is run on 2014-09-15, it searches for data from 2014-04-01 to 2014-06-30.
Last 7 Days	The report searches for data from the last 7 days.
Last 30 days	The report searches for data from the last 30 days.

4 In **Share with**, select one of the following options:

- **Only me**: The report is private. Noone else can see the report.
- **Public**: The report is available to all users who have access to Reporter.
- **Users/Groups**: The report is available to selected users or user groups.

5 If you selected **Users/Groups**, specify each user or group of users with whom to share the report:

a In the text field that appears, type the first letter of the user's first name or the group name. A list displays all the user and group names that begin with the letter you typed as shown in Figure 2.3-11.

b Select a name.

c Repeat the previous steps to add names as needed.

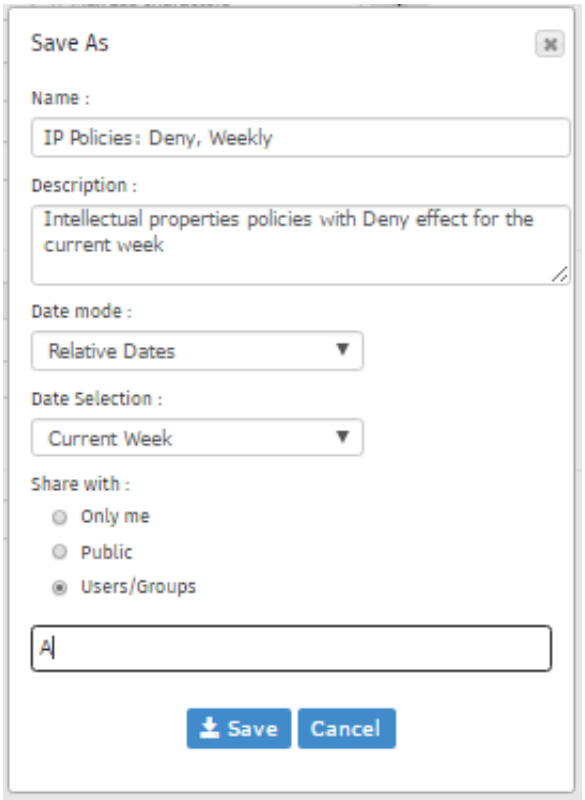


Figure 2.3-11: Selecting users with whom to share a report

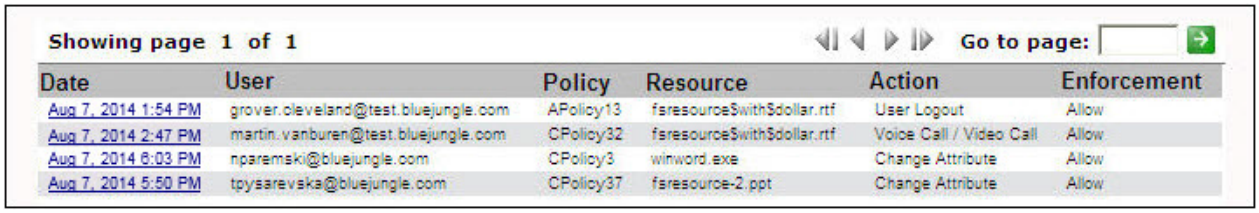
6 Select **Save**. The report appears in the **Saved Reports** pane.

Sample reports

This section presents examples of different report formats, all representing a small set of event data returned by the same report query. The query requests all events by all users on all resources for one week (August 3, 2014 to August 9, 2014). By comparing the examples, you can get a better understanding of the way the different formats can be used to highlight different aspects of the same data.

Table

Figure 2.3-12 shows a small set of policy activity data displayed in a table.



Showing page 1 of 1 Go to page:

Date	User	Policy	Resource	Action	Enforcement
Aug 7, 2014 1:54 PM	grover.cleveland@test.bluejungle.com	APolicy13	fsresource$with$dollar.rtf	User Logout	Allow
Aug 7, 2014 2:47 PM	martin.vanburen@test.bluejungle.com	CPolicy32	fsresource$with$dollar.rtf	Voice Call / Video Call	Allow
Aug 7, 2014 6:03 PM	nparemski@bluejungle.com	CPolicy3	winword.exe	Change Attribute	Allow
Aug 7, 2014 5:50 PM	tpysarevska@bluejungle.com	CPolicy37	fsresource-2.ppt	Change Attribute	Allow

Figure 2.3-12: Data presented in a table

This report reflects the following:

- Four policy enforcement events were logged—that is why there are four rows in the table.
- As the Date column shows, all four events occurred on the same day.
- Each event was triggered by a different user, as shown by the four different values in the User column.
- Each event represents a different policy, as shown by the four different values in the Policy column.
- Only three resources were involved. The first two rows show enforcement of two different policies by two different users, but both were connected to the same RTF document. This means that document is covered by (at least) two policies concurrently—not an unusual situation.
- The enforcements were triggered by three different user actions, as shown by the values in the Action column. The third and last instances were caused by different users doing the same thing—trying to change a document's attributes.
- In all four cases, the enforcement decision was Allow, as shown in the last column.

Keep these details in mind as you analyze the data in the following charts.

Group by policy chart

As discussed, four policies are reflected in the table above. When the report is re-run with **Bar Chart** selected in the **Report Type** field, and **Group By Policy** selected in the **Show** field, Reporter generates a bar chart with four bars, one per policy. Each bar is the same height, indicating one enforcement event per policy, as shown in Figure 2.3-13.

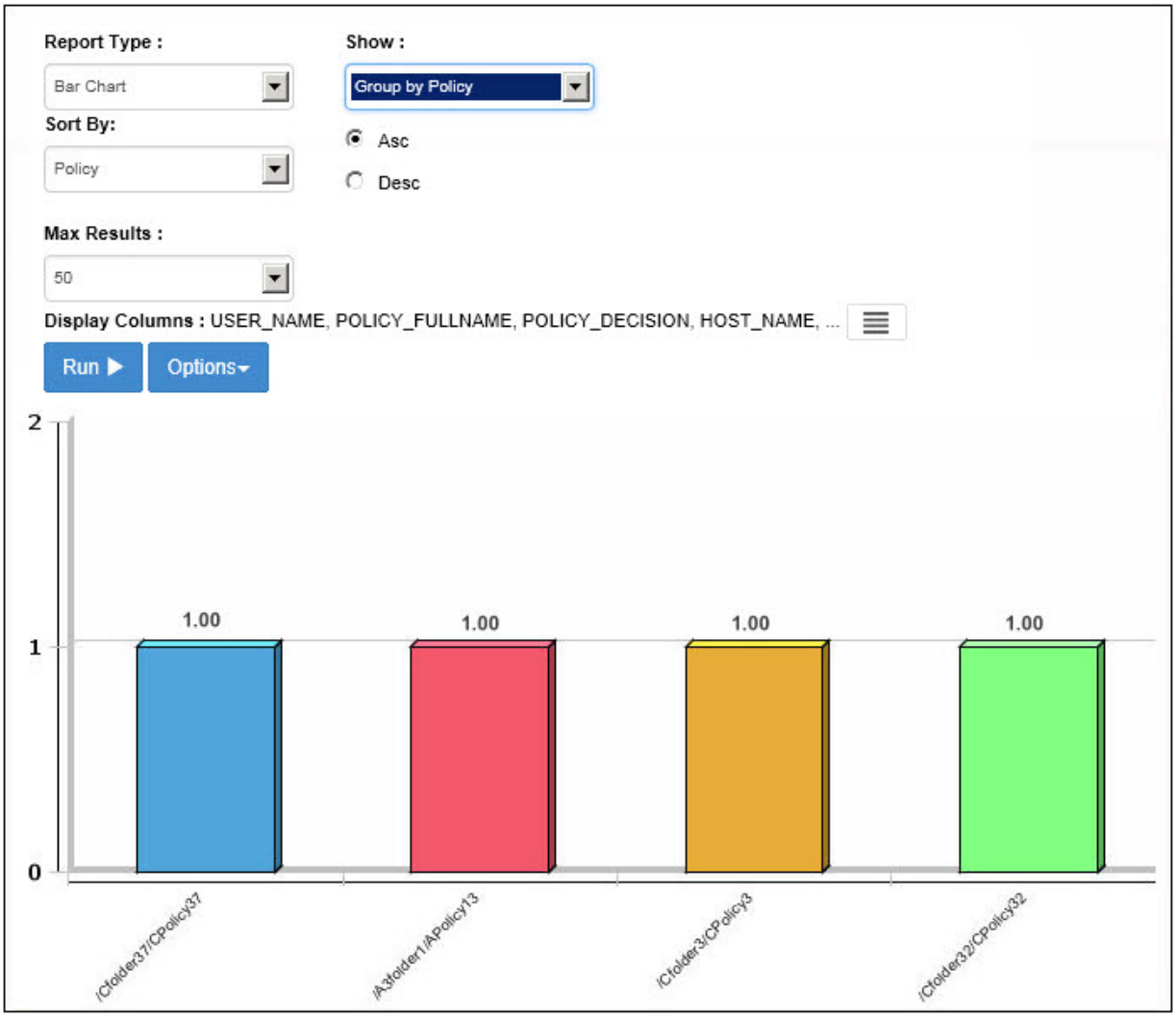


Figure 2.3-13: Group By Policy chart

Grouping events by policy is useful for identifying policies that are being triggered with unexpected frequency, which may be an indication that they are improperly designed and cover users, resources or actions that they should not. It can also be an indication of concentrated efforts at unauthorized data access. To examine the latter possibility, it is often helpful to switch to the **Group by User** option, to focus on who is performing the activity.

Group by user chart

When the same data is grouped by user, and the bar chart is selected, the chart is generated, as shown in Figure 2.3-14. As noted previously, the four policies were each triggered by a different user, and so the graph shows four bars—each representing one user. Each is labeled with a username. In this example, the bars are the same height, since each of the four users triggered a policy once.

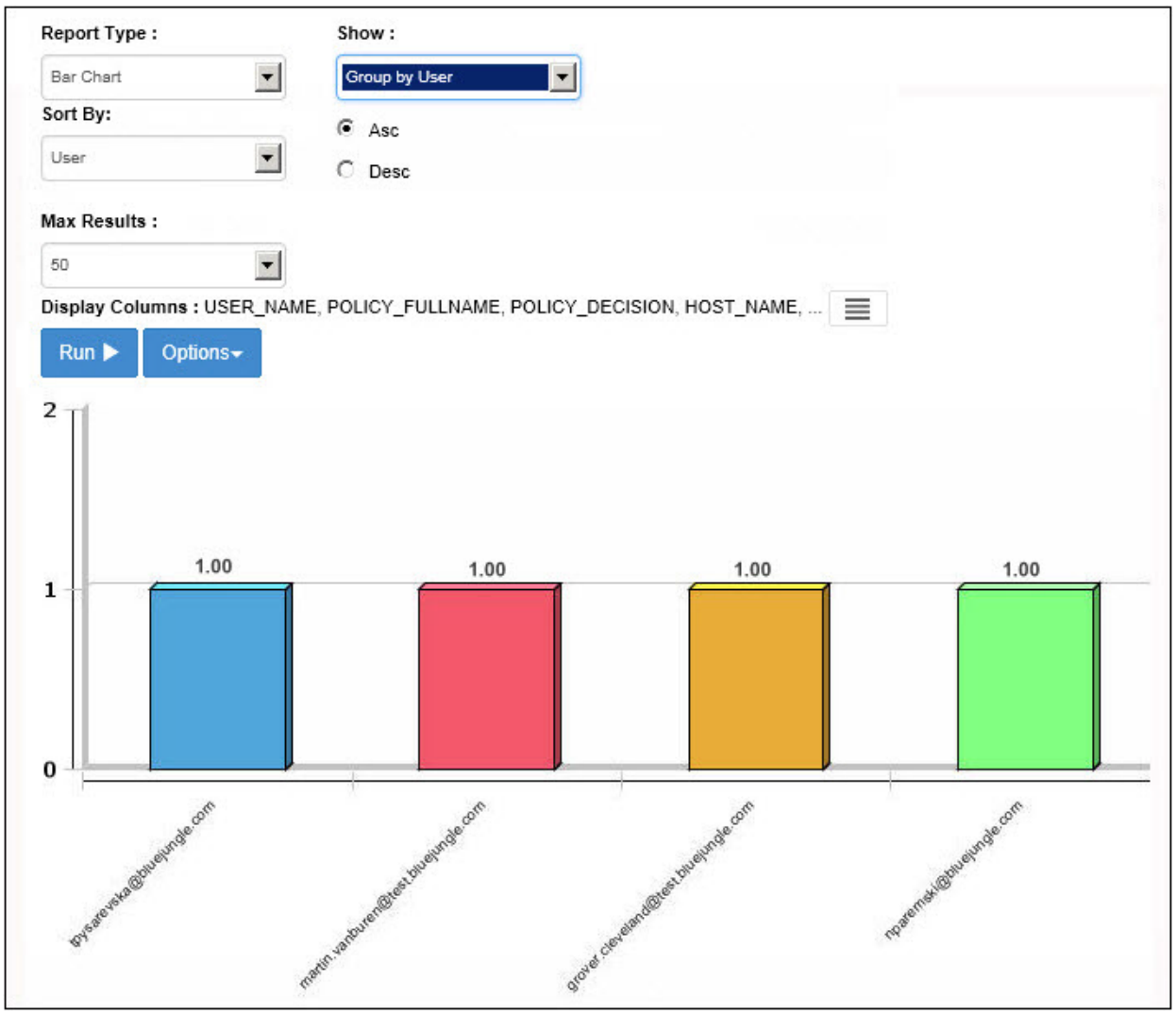


Figure 2.3-14: Group By User chart

In a typical live environment, there is little difference among users' rates of triggering policies. If a user shows a significantly disproportionate rate of triggering policies, especially Deny policies, the situation merits closer investigation. Figure 2.3-15 shows an example of how such a case might look in a chart where data is grouped by user.

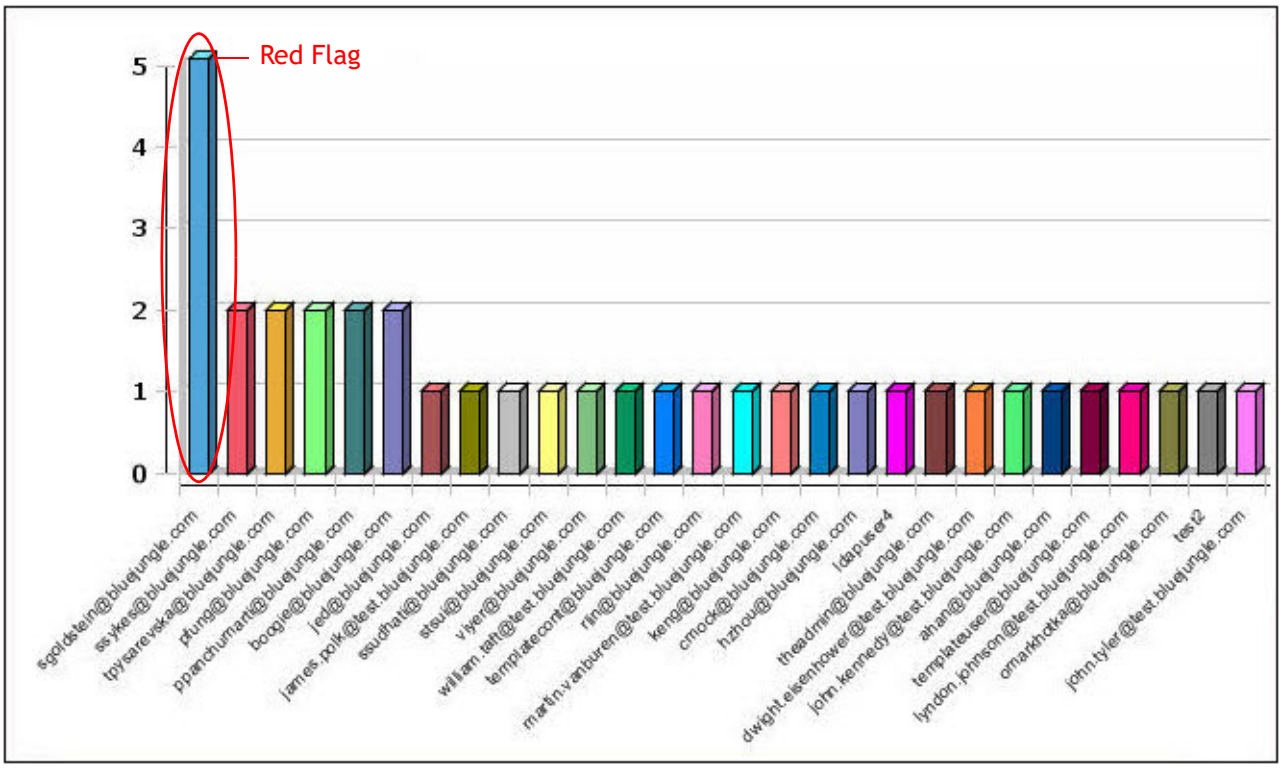


Figure 2.3-15: Group by user chart—red flag

Group by resource chart

When the same data is grouped by resource, the chart shows the extent of specified events—in this case, policies being triggered—per each individual resource covered by the report. In our case, of the four events, two involved the same RTF file (shown in blue) and the other two, one file each (red and orange). This means three resources total, as shown in Figure 2.3-16.

Because policies often cover large numbers of individual documents or other resources, grouping by resource is only helpful when the number of events has already been narrowed down to a smaller set by various report filters, such as policies or users. A pie charts is ideal here, because in the context of resource use, the *relative* access activity regarding some single file or other resource as compared to all others, is generally of more interest than any *absolute* number of instances of access.

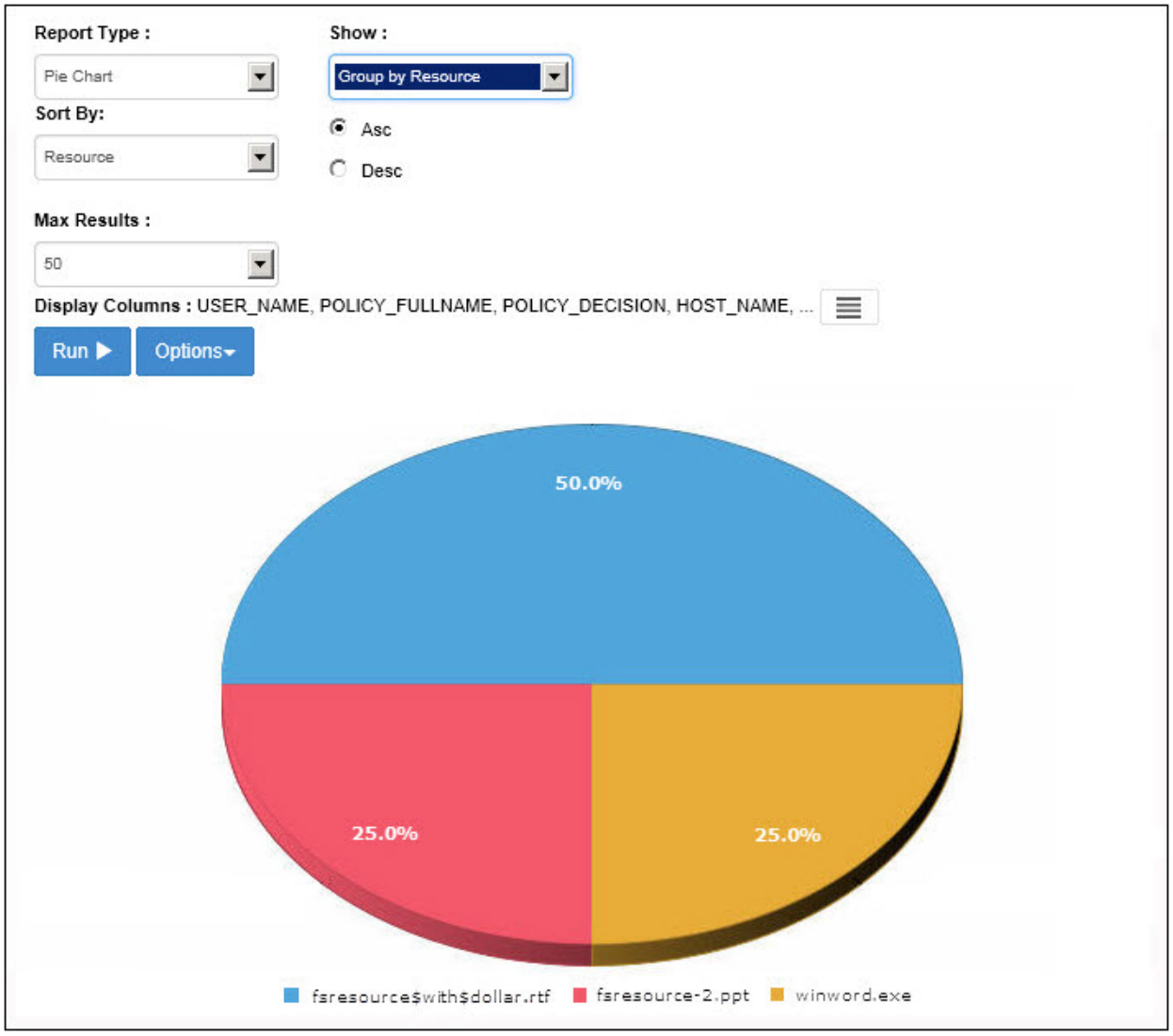


Figure 2.3-16: Group by resource chart

Group by time chart

The **Group by Month** and **Group by Day** options group all events by month and day, respectively. In our example, since all four events occurred on the same day, the chart shows only one bar, labeled 07 Aug 2014, as shown in Figure 2.3-17. The height of the bar indicates that four incidents occurred on that date. Because all the events occurred on the same day for a reporting period of one week, the chart appears the same whether the **Group by Month** or **Group by Day** option was selected. It makes sense to select the **Group by Month** option only if the report period you specify spans more than one month.

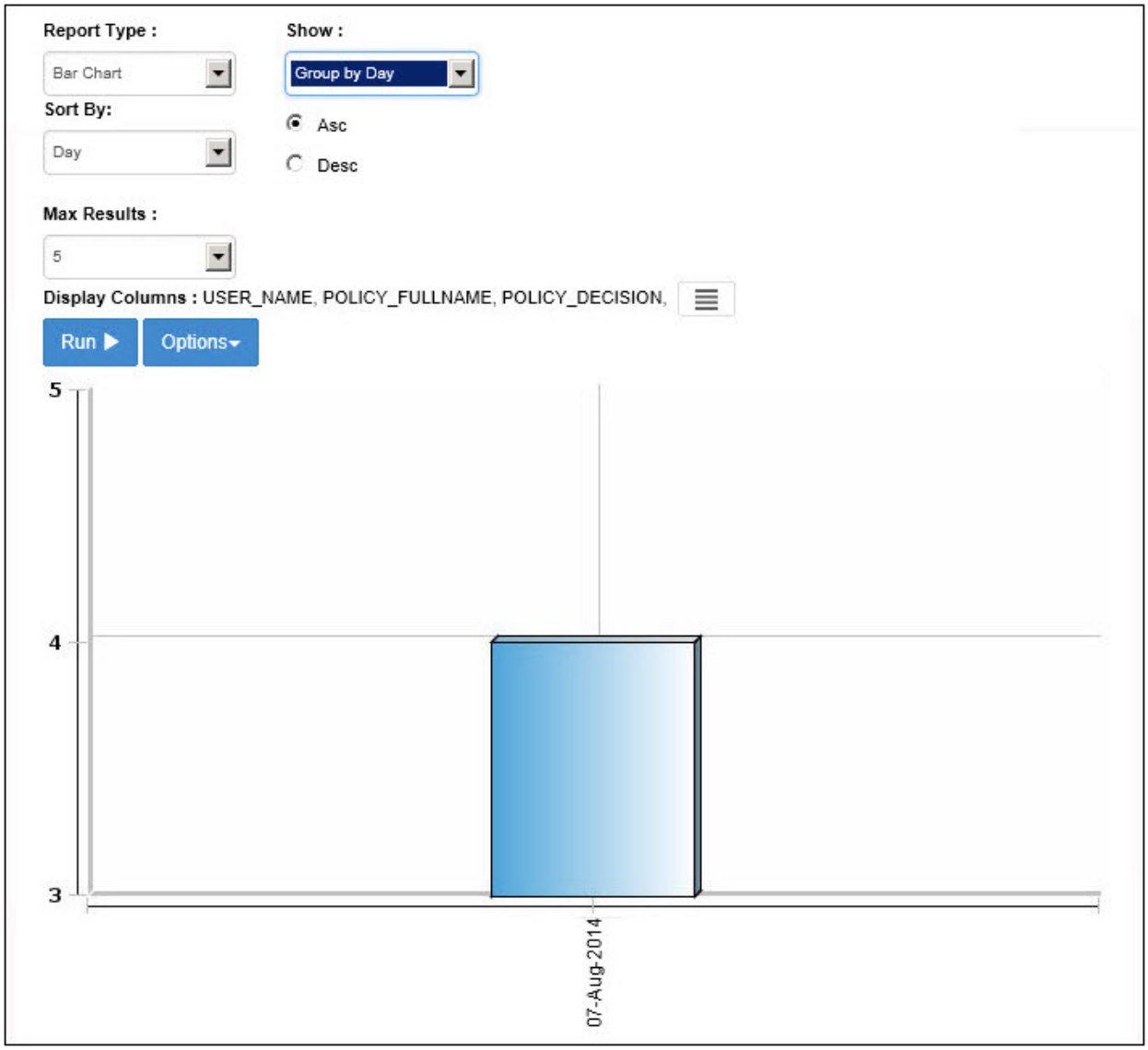


Figure 2.3-17: Group By Day chart

In a live environment, this format is particularly useful in identifying time-related enforcement spikes, which are similar to the user-related red flags. Spikes in policy enforcement at specific times, such as weekends or the end of a quarter (especially with regard to Deny policies) can indicate intentional efforts at unauthorized access to sensitive resources or to financial documents prepared at the end of each quarter.

Report anomalies

Reports can show unexpected results at times, which may seem to reflect incorrect behavior of policies, but in fact are a result of some of the ways Windows works internally, or of inherent statistical issues.

Implied actions

When Windows Explorer handles file operations, it sometimes performs several actions sequentially in a way that is not visible to users. For example, when you run Copy on a file, Explorer first opens the file, then reads the content, then writes a copy to the new location. Similarly, when you print a file, Explorer must first open the file, then send it to the print server. In fact, most of the actions you deal with in policies also involve Open in this way, as what might be called an *implied action*.

Because Open is a very common implied action, you may see some reporting anomalies whenever you deploy several policies, one of which involves a Deny Open or Allow Only Open effect. For example, suppose you have a Deny Open policy deployed, then deploy a second policy with a Deny Print effect, test it for a while, and run a report on the test activity. The test behavior might seem correct—printing is blocked, as intended—but the report does not show any instances where Print actions were denied; instead it shows many cases of Open actions being blocked. This can seem like an error in the policy or in the report, but it actually reflects the fact that, because the actions implicitly triggered the Deny Open policy, they were blocked by that rather than by the Deny Print policy.

Note that if you encounter this kind of seeming anomaly regarding actions in your reports, it is likely that some kind of implied action (involving Open, specifically) is the cause.

Preview activity

When you preview document-type components, the system performs a Read action on the underlying documents to display information about them in Preview pane. This may lead to unexpected reporting results if, for example, you have deployed policies that prohibit users from reading these document resources. Even though the policy is correctly blocking this activity by users, your report might show instances of Read actions—those performed in the course of Preview. This can be confusing at times, since there is no way Reporter can distinguish between actions performed internally by the system and actions performed by users.

2.4 Working with monitors and alerts

This section describes how to use Reporter to create policy monitors to obtain notifications of policy enforcement activities that meet predefined criteria. Reporter is the console used to monitor and report on policy compliance, gather statistics about document usage, and investigate any suspected incidents of information mishandling. Reporter is used by administrators, IT staff, managers, executives, auditors, and other authorized personnel.

Topics in this section

Monitoring functionality and permissions

Monitors and alerts can provide continuous coverage of key policy enforcements in an enterprise, as well as a notification system that warns administrators when action is required. In Reporter, you use the *Monitoring* tab to create policy monitors and to view alerts triggered by those monitors.

The functionality available to you depends on the rules established in the *Delegated Administration* section the console.

When you click the *Monitoring* tab, the *Alerts Overview* page appears. It displays a list of alerts, if any, in the order they were raised, with the latest on top.

If there are alerts, you can take the following actions:

- Filter alerts by specified criteria, including time period, monitor, and tag name and value
- View alerts aggregated by tag, monitor, or time, and presented in chart format
- Dismiss any alert from the list
- Delete any alert from the list

Two subtabs appear below Monitoring: *Alerts* and *Monitors*. Click **Monitors** to view the list of defined monitors, if any, and to create a monitor.

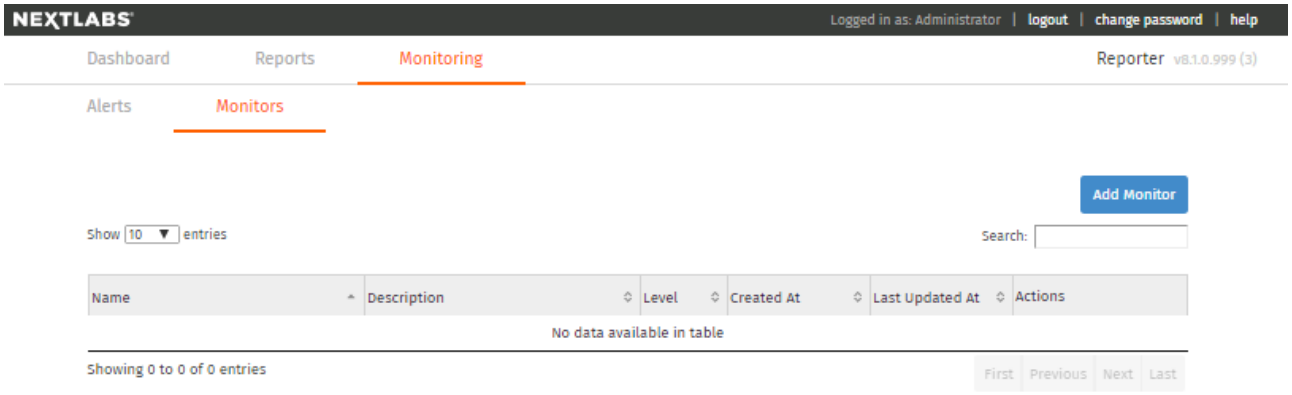


Figure 2.4-1: Monitoring tab and subtabs

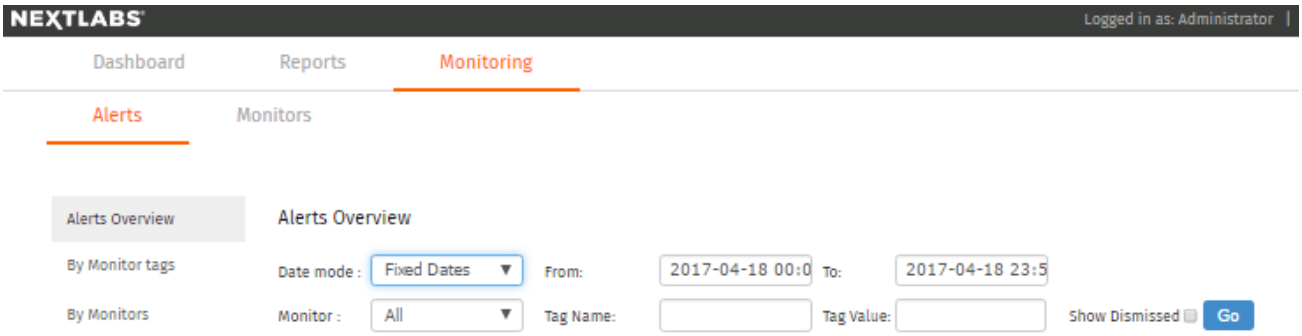


Figure 2.4-2: Alerts Overview page

As Figure 2.4-2 shows, two subtabs appear below Monitoring: **Alerts** and **Monitors**. Click **Monitors** to view the list of defined monitors, if any, and to create a new monitor.

Creating a monitor

Creating a monitor is similar to creating a report. Both access policy activity data from the Activity Journal. You define the following properties for a monitor:

- The criteria, or filters, that specify what policy activities to watch. As with a report, a monitor can filter activity data by user, resource, user action, policy, application, and host. For example, you can create a monitor to watch all policy activities where downloads of documents classified as Confidential are denied.
- The conditions that determine when an alert is raised. For example, you can create a monitor to raise an alert when the activity specified above occurs more than 10 times in one day, or when more than 10MB of confidential documents have been downloaded in one week.

- The type of alert—an email notification, or an alert displayed in the **Alerts Overview** page, or both.
- A monitor tag. Although tags are optional, they are useful for organizing monitors and the alerts they generate. In the *Alerts* tab, you can filter and analyze alerts by tag.

To create a monitor, perform the following steps. Note that steps or properties are optional unless described as required.

1 In the *Monitors* tab, click the **Add Monitor** button. The **Create Monitor** dialog appears, with the *General* tab open.

2 In the *General* tab, define the general properties of the monitor.

- In **Name**, type a name for the monitor. This property is required.
- In **Description**, provide a description of the monitor.
- In **Duration**, select a value from the drop-down list. Duration is the time period that the monitor considers when evaluating filters and other conditions that you specify for the monitor. This property is required, and the default is Today. See Specifying the Duration.
- In **Group By**, select a grouping option from the drop-down list. This property specifies whether the condition specified in the next property, **Aggregators**, applies at the selected group level, or to all the policy activity records covered by the monitor (the default).
- In **Aggregators**, specify a file size or number of records threshold that the monitor must meet or exceed before raising an alert. This property is required. See Specifying the file size or records criteria.
- In **Alert**, select how you want to receive alerts raised by the monitor. By default, all alerts are listed in the *Alerts Overview* Tab. In addition, you can elect to receive an email alert. If you select the email option, enter an email address.
- In **Level**, select a value from the drop-down list. You can use level to indicate the priority or seriousness of the alert generated by the monitor. In the *Alerts Overview* tab, you can sort alerts by level, so it is useful to assign a meaningful level to each monitor. By default, all alerts are set to L1.
- In **Message**, type a message that you want the alert to display. For example:

```
Access to ITAR resources was denied more than 10 times today
```

Depending on the option or options you selected for the **Alert** property, this message appears in the *Alerts Overview* tab, or in an email, or both.

3 Click the **Filters** tab to define the criteria that determine the policy activities to monitor. The filter options are the same as the ones you can specify in a report query. You can filter on the following: Action, User, Resource, Policy, and Other attributes.

For details about defining filters, see Filtering by user, resource, and policy on page 165 and Filtering by action.

4 Click the **Tags** tab to create one or more tags for the monitor.

- In **Name**, type a name for the tag.
- In **Value**, type a value for the tag.
- Click the + button to add the tag to the monitor.

See Defining a tag.

5 Click the **Access Control** tab to grant other users access to the monitor. Other users can only view the definition of the monitor; they cannot modify it.

6 In **Share with**, select one of the following options:

- **Only me**: The monitor is private. No one else can see this monitor or the alerts it generates.
- **Public**: The monitor, and any alerts it generates, can be viewed by all users who have access to Reporter.
- **Users/Groups**: The monitor, and any alerts it generates, can be viewed by selected users or user groups. If you select this option, specify each user or group of users with whom to grant access to the monitor. In the text field that appears, type the first letter of the user's first name or the group name. Select a name from the list that appears. Repeat the previous steps to add names as needed.

7 Click **Save**. The monitor is activated. It appears in the list of monitors in the *Monitors* tab, where you can edit, deactivate, or delete it.

Specifying the Duration

The **Duration** property defines the time period that the monitor covers when evaluating filters and other conditions that you specify for the monitor. For example, to create a monitor that raises an alert when Policy01 denies access to documents classified as ITAR (International Traffic in Arms Regulations) more than 20 times *in a week*, you would select **Current Week** as the duration.

As long as the monitor remains active, it watches for policy activities that meet the filter conditions each week, and raises an alert if the number of records (policy activities) threshold of 20 is exceeded for that week.

While it is typical to use monitors to track current policy activities and receive alerts when certain conditions occur, monitors can also be used to evaluate past activities. For example, a comparison of historical data with current data can reveal trends, such as whether an organization's information controls are more or less effective from one period to the next. A monitor can evaluate past data up to 90 days prior, which is, by default, the amount of time data is stored in the Activity Journal.

By default, a monitor can only evaluate data that is logged *after* the monitor's creation date and time. For example, if you create a monitor and specify Last Week as the duration, the monitor cannot access data from the previous week until a week later when a week's data has accumulated. If you want to evaluate past activities for a specific time period, plan ahead and create the monitor before that time.

Table 2.4.1 describes the time periods you can select for **Duration**.

Table 2.4.1: Duration values

Value	Description
Today	The monitor evaluates data for the current day. Use Today for critical policies whose activities you want to monitor daily for unusual or unacceptable levels of enforcement.
Current Week	The monitor evaluates data for the current week. A week is from Sunday to Saturday.
Current Month	The monitor evaluates data for the current month.
Current Quarter	The monitor evaluates data for the current quarter.
Yesterday	The monitor evaluates data from the previous day.
Last Calendar Week	The monitor evaluates data in the previous week (Sunday to Saturday).
Last Calendar Month	The monitor evaluates data in the previous month (first day to last day in the month).
Last Calendar Quarter	The monitor evaluates data in the previous quarter (first day to last day in the quarter).
Last 7 Days	The monitor evaluates data from the last 7 days. Each day the monitor is active, the window of 7 days moves up by one day.
Last 30 days	The monitor evaluates data from the last 30 days. Each day the monitor is active, the window of 30 days moves up by one day.

Specifying the file size or records criteria

The **Aggregator** property requires at least one condition—either file size or number of records—to apply to the activity data that match the specified duration and filters. A monitor can include both conditions.

Number of Records is the number of policy activities that meet the duration and filters defined for the monitor.

The other condition you can specify involves file sizes. In the previous monitor examples, the aggregator conditions are applied across all the activity records that match the specified duration and filters. If, however, the monitor specifies a grouping option in the **Group By** property, an aggregator condition applies at the selected group level. Suppose you want to change the previous monitor to do the following: Raise an alert when Policy02 has allowed the downloading of ITAR documents totaling more than 10MB, *by any one user*, in a week. In **Group By**, you would select **Group by User**. The File Size condition would then apply to user groups, rather than to all activity records.

Defining a tag

Assigning a tag is optional, but useful for organizing alerts raised by the monitor, as well as for gathering statistics, by tag, on those alerts. In the Alerts page, you can view alerts that are grouped and aggregated by monitor tags. See Alerts by monitor tag.

When deciding if a monitor should be tagged, and what tag or tags to create, think about the ways you want to organize or categorize alerts. For example, you can create a tag that reflects

the purpose of the monitor, that is, the type of policies, the application system, the resources, or the groups of users, being monitored. You can also create a tag that indicates the type of risk or risk level associated with the policy activities being monitored.

In the example above, two tags are defined for the monitor, which enables the alerts triggered by the monitor to be organized and analyzed in two ways.

- The SAP tag is set to the value DMS. This indicates that the monitor watches for policies that enforce information controls on the DMS application module of an SAP system. There are probably other monitors that watch policies associated with other modules of an SAP system, such as PLM or EasyDMS. By tagging a monitor as SAP, you can obtain statistics on the number of alerts triggered for SAP in general, as well as for each SAP application module.
- The ITAR Policies tag is set to the value Access Control. In addition to watching for policy enforcement on an SAP DMS system, the monitor also watches for policies that control access to ITAR data. Presumably there are other similar monitors that watch policies that control the storage, use, or distribution of ITAR data. By assigning an ITAR Policies tag, you can obtain statistics on the number of alerts triggered for ITAR policies in general, and for each type of control governing ITAR policies.

Deactivating and deleting monitors

When you create and save a monitor, it is activated automatically, and stays in effect until you deactivate or delete it. Deactivating or deleting monitors are common tasks during the testing phase. You may want to deactivate a monitor to temporarily suspend it. You can reactivate a monitor at any time. If a monitor no longer serves a purpose, delete it. When you delete a monitor, all alerts it triggered are also deleted.

To deactivate or delete a monitor, click **Monitors** to open the *Monitors* tab. Select the monitor, and in the Action column, select **Deactivate** or **Delete**. If your system contains a large number of monitors, you can use **Search** to find a monitor. You can search for any value in any of the columns (Name, Description, Created At, and so on) by typing a full or partial value.

Viewing alerts

The *Alerts Overview* tab provides a summary list of alerts in a table. By default, the table displays all alerts raised by all monitors in the current day. The alerts are sorted by the time they were raised, starting with the latest. By default, ten alerts are shown on each page. You can increase the number of alerts to display per page by selecting a number in **Show 10 entries**. Use the navigation links on the bottom right to browse through all the alerts.

Sorting alerts

By default, alerts in the Alerts Overview table are sorted by date and time values in the Raised At column. You can select a different column—Level, Monitor Name, or Alert Message—on which to

sort by clicking the column name in the table header. By also clicking the column header, you can change the sorting order of a column from descending to ascending order, and vice versa.

Filtering alerts

You can filter the list of alerts to view only the ones of interest. You can filter by dates (specific or relative), monitor name, tag name, or tag value.

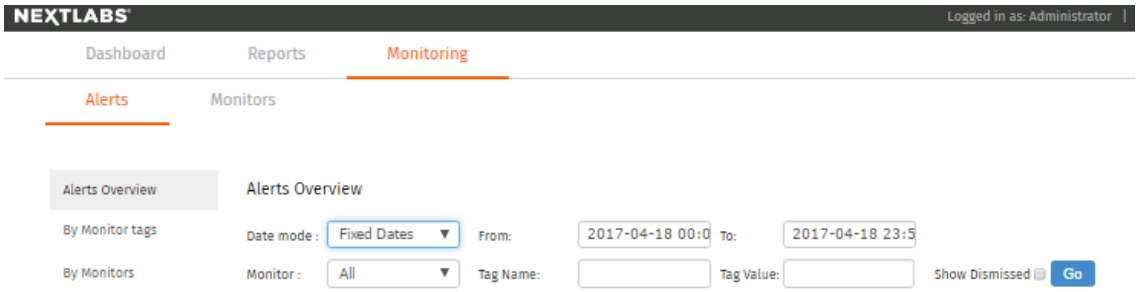


Figure 2.4-3: Filtering alerts

Filtering by specific dates or times

A specific time period can be any span of time. For example, you can define a filter that selects data for a specific week or day. You can narrow the data to a range of hours, or even minutes, in a specific day. To display alerts triggered in a specific time period, perform the following steps.

Procedure

1. In *Date* **mode**, select **Fixed Dates**.
2. In **From** and **To**, specify the start date/time and end date/time, respectively, of the time period in which the alerts you want to view, occurred. Click in the field to choose a date and/or time from the calendar.
3. Click **Go**.

Filtering by relative dates

Relative dates are the dates that are relative to the current date. For example, you may want to view alerts that were triggered yesterday or this week. You can display alerts by relative dates.

Procedure

1. In *Date* mode, use the default option **Relative**.
2. In **Date Selection**, select a time period from the drop-down list. For descriptions of each value, see Duration values.
3. Click **Go**.

Filtering by monitor

You have the option of viewing alerts triggered by all monitors (the default) or by a specific monitor. In **Monitor**, select All, or one of the monitors in the drop-down list.

Filtering by tag name and tag value

As described in Creating a monitor, you can create one or more tags for each monitor. Each tag is a name-value pair. You can filter alerts by either name or value, or both.

Displaying alert details

You can view all the policy enforcement events associated with the alert. To display this information, select **Details** in the Action column.

Deleting and dismissing alerts

The system retains all alerts until you delete them. If there are alerts you no longer need to see or include in analyses, you should delete them. For each alert to delete, select **Delete** in the Action column.

If you want to keep an alert in the system for statistical reasons, but want to exclude it from the table in Alerts Overview, select **Dismiss** in the Action column. To re-display alerts that you have dismissed, select **Show Dismissed** above the table.

Analyzing alerts

In addition to displaying the complete list of alerts, Reporter provides the option of viewing and analyzing alerts in the following ways:

- Alerts by monitor tag
- Alerts by monitors
- Alerts by time

Alerts are grouped by tag, monitor, or time, aggregated, and presented in charts. To view alerts in one of these formats, in the *Alerts Overview* tab, select an option from the menu on the left, as shown in Figure 2.4-4.

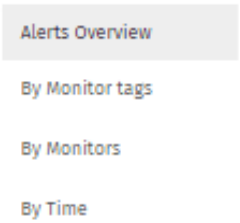


Figure 2.4-4: Alerts menu

Alerts by monitor tag

When you select the option to view alerts by monitor tag, Reporter groups alerts by tag name, and presents the data in a treemap, as shown in Figure 2.4-5. Each rectangle in a treemap represents a tag, and the rectangle sizes let you compare at a glance the number of alerts associated with each tag.

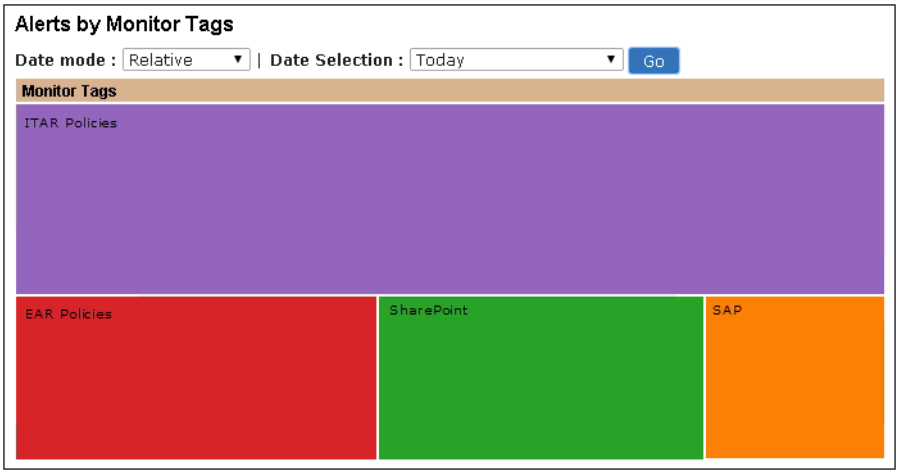


Figure 2.4-5: Alerts by Monitor Tag treemap

To see the distribution of alerts by tag *value* for a particular tag, click the rectangle representing the tag. Figure 2.4-6 shows the distribution of alerts, by tag value, for the ITAR Policies tag.

To go back to the top-level treemap, click the **Monitor Tags.ITAR Policies** title bar. To display the list of alerts associated with a tag value, click the corresponding rectangle.

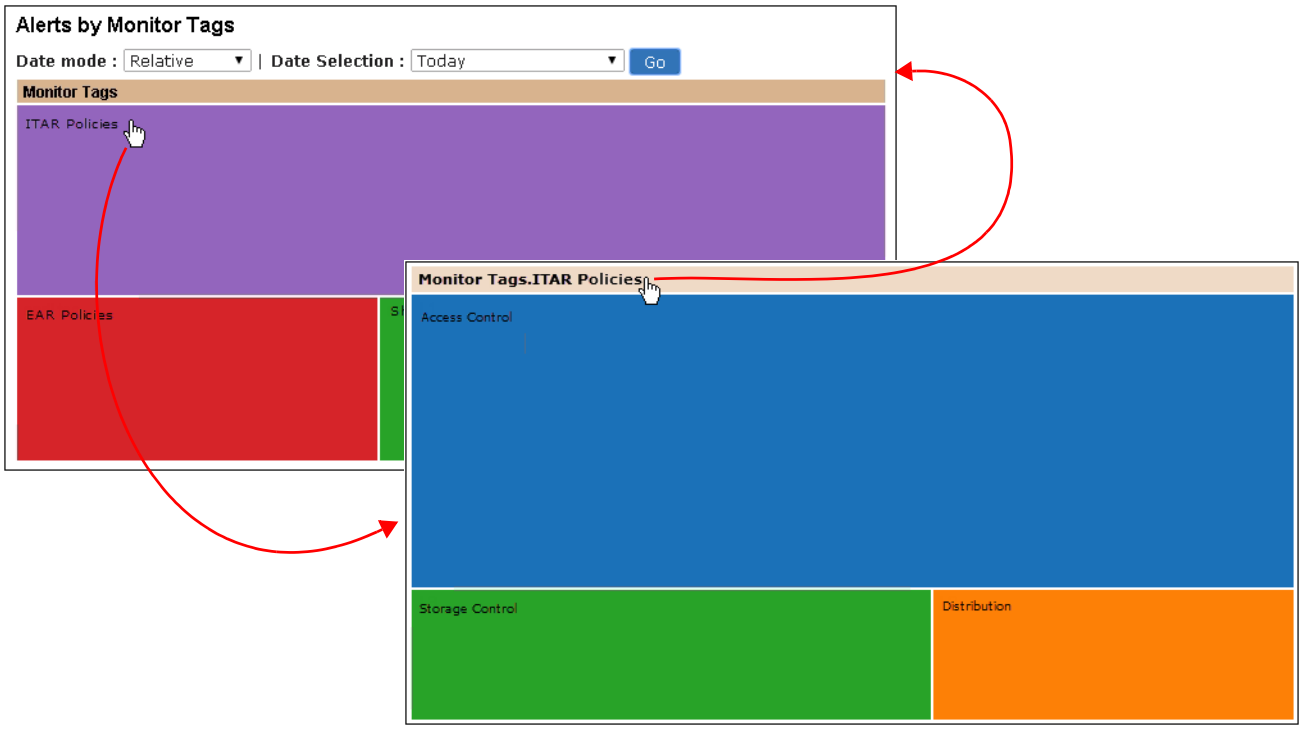


Figure 2.4-6: Clicking a tag to view the distribution of alerts by tag value

By default, the treemap shows alerts for the current day. To change the time period, in **Date mode**, select the value you want. See Filtering by specific dates or times and Filtering by relative dates.

Alerts by monitors

When you select the option to view alerts by monitor, Reporter groups alerts by monitor name, and presents the data in a bar chart, as shown in Figure 2.4-7. The labels in the x-axis show the monitor names.

By default, the chart shows the number of alerts for each monitor, for the current day. To change the time period, in **Date mode**, select the value you want. See Filtering by specific dates or times and Filtering by relative dates.

To display the list of alerts associated with a monitor, click the corresponding bar.

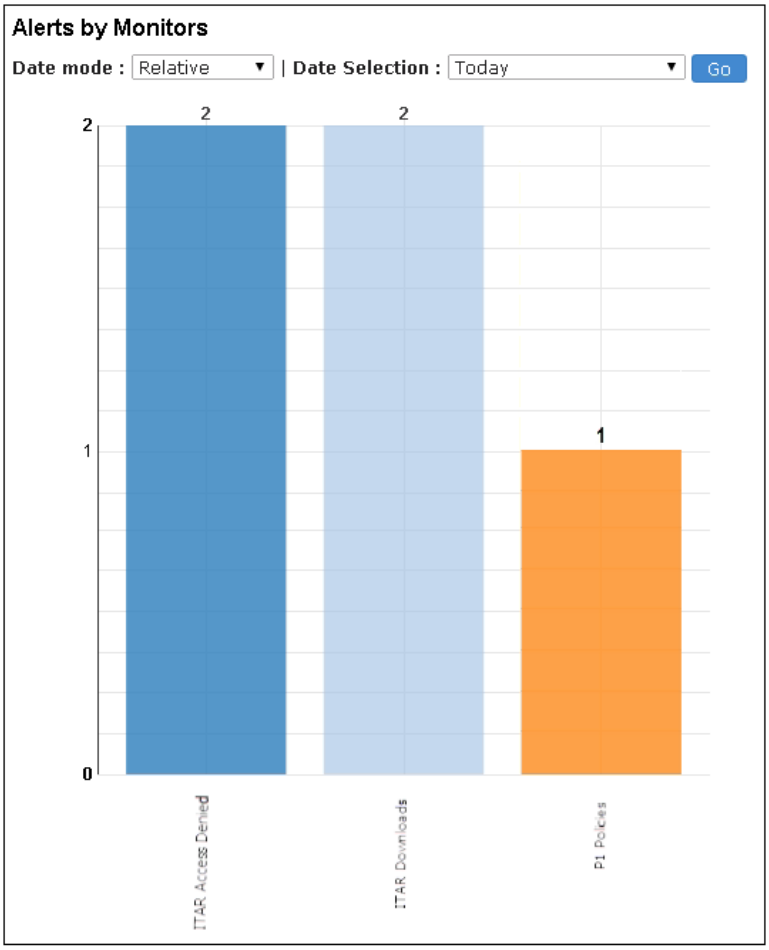


Figure 2.4-7: Alerts by Monitor chart

Alerts by time

When you select the option to view alerts by time, Reporter groups alerts by the time period that you specify, and presents the data in a bar chart; if you don't change the default settings, the time period is the current day. To change the time period, in **Date mode**, select either Fixed Dates or Relative. See Filtering by specific dates or times and Filtering by relative dates.

Figure 2.4-8 shows two chart examples. The chart on the left shows the number of alerts for the current day (the default), and the chart on the right shows the number of alerts per day for the current week. In the second chart, **Date mode** is set to **Relative**, and **Date Selection** is set to **Current Week**.The labels in the x-axis show the dates. Click a bar in the chart to view the alerts for that day.

Figure 2.4-8: Two charts, one displaying alerts for the current day, the second for the current week

In the examples above, alerts are grouped by day. If **Date Selection** is set to Current Quarter or Last Quarter, alerts are grouped by month.

3.5 Using Audit Logs

This section describes how to use the *Audit Logs* tab to view activity logs.

About Audit Logs

In Reporter, you can use the *Audit Logs* tab to view audit logs. The audit logs include the creation, modification, and deletion of the following entities:

- Action
- Application User
- Delegate Admin Rule
- Policy
- Policy Model
- Report
- Resource
- Subject

Audit logging is an automated function, and does not require any configuration.

Viewing Audit Logs

Procedure

1 Log in to Control Center with an account that has permission to view reports, such as the superuser Administrator account.

2 In the Control Center console, navigate to **Reports > Audit Logs**.
Audit Logs pane

3 In the Audit Logs pane, define the query criteria.

Table 3-5-1: Audit logs query criteria

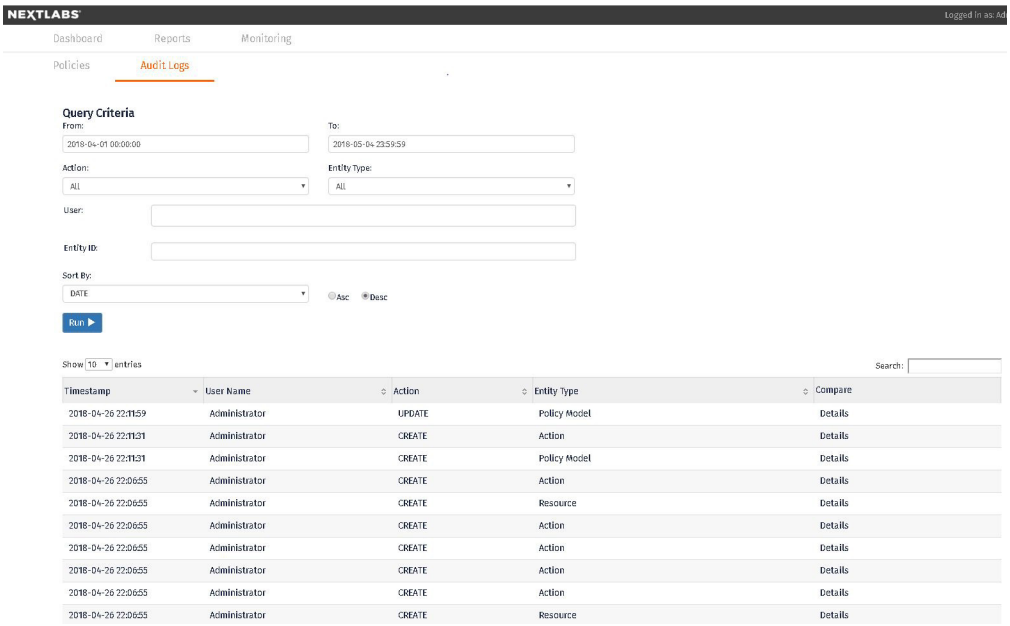
Field	Description
From and To	The start date and time, and end date and time, respectively, of the time period this report covers. Click in the field to choose a date and time from the calendar.
Action	The actions you want to show the log for. You can select All, Create, Update, and Delete.

Table 3-5-1: Audit logs query criteria

Field	Description
Entity Type	The type of entity you want to show the log for. You can select one of the following entities: • All - Selects all the entities • Action • Application User • Delegate Admin Rule • Policy • Policy Model • Report • Resource • Subject
User	If you define a user, or several users, the log only shows the activities from that user(s). You can type the first letter of the username, and select from the drop-down list.
Entity ID	The value of the ENTITY_ID Column in ENTITY_AUDIT_LOG table. All entities such as policy model resource, subject/resource/action components, policies, users, and delegation policies have entity IDs.
Sort By	You can sort the result of your query by date, entity type, or action.
Asc or Desc	You can sort the result of your query by the order it appears in. You can select ascending or descending from the drop-down list.

4 Click **Run**.

5 The query results appear in a table at the bottom of the page.



You can show the number of results you want to see, and search within the results.

6 To view entity details, click **Details**. For example, entity details for CREATE and DELETE operations are displayed as shown in Figure 3.5-1 and entity details for UPDATE operations are displayed as shown in Figure 3.5-2.

Figure 3.5-1: Entity details for CREATE and DELETE operations

Figure 3.5-2: Entity details for UPDATE operations

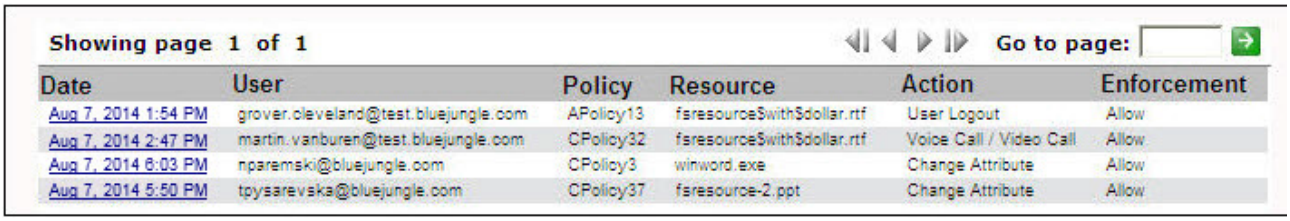
2.6 Sample reports

This section presents examples of the available report formats, all representing a simple set of event data returned by the same report query. The query requests all events by all users on all resources. By comparing the examples carefully, you can get a better understanding of the way the different formats can be used to highlight different aspects of the same data.

Topics in this section

Event Details reports

Figure 2.6-1 shows the Event Details report on a small set of policy activity data.



Showing page 1 of 1 Go to page:

Date	User	Policy	Resource	Action	Enforcement
Aug 7, 2014 1:54 PM	grover.cleveland@test.bluejungle.com	APolicy13	fsresource$with$dollar.rtf	User Logout	Allow
Aug 7, 2014 2:47 PM	martin.vanburen@test.bluejungle.com	CPolicy32	fsresource$with$dollar.rtf	Voice Call / Video Call	Allow
Aug 7, 2014 6:03 PM	nparemski@bluejungle.com	CPolicy3	winword.exe	Change Attribute	Allow
Aug 7, 2014 5:50 PM	tpysarevska@bluejungle.com	CPolicy37	fsresource-2.ppt	Change Attribute	Allow

Figure 2.6-1: Event Details Report

The Event Details report reflects the following:

- Four instances of policy enforcement were logged—that is why there are four rows in the table.
- As the Date column shows, all four instances occurred on the same day.
- Each was triggered by a different user, as shown by the four different values in the User column.
- Each represents a different policy, as shown by the four different values in the Policy column.
- Only three resources were involved. The first two rows show enforcement to two different policies by two different users, but both were connected to the same RTF document. This means that document is covered by (at least) two policies concurrently—not an unusual situation.
- The enforcements were triggered by three different user actions, as shown by the values in the Action column. The third and last instances were caused by different users doing the same thing—trying to change a document's attributes.

- In all four cases, the enforcement result was Allow, as shown in the last column.

Bear these details in mind as you analyze the Group by Details reports below.

Group By Policy reports

When you run a report with Show Group By Policy selected, the result a bar chart with four bars, one per policy. For each, the height is the same, indicating one enforcement instance apiece.

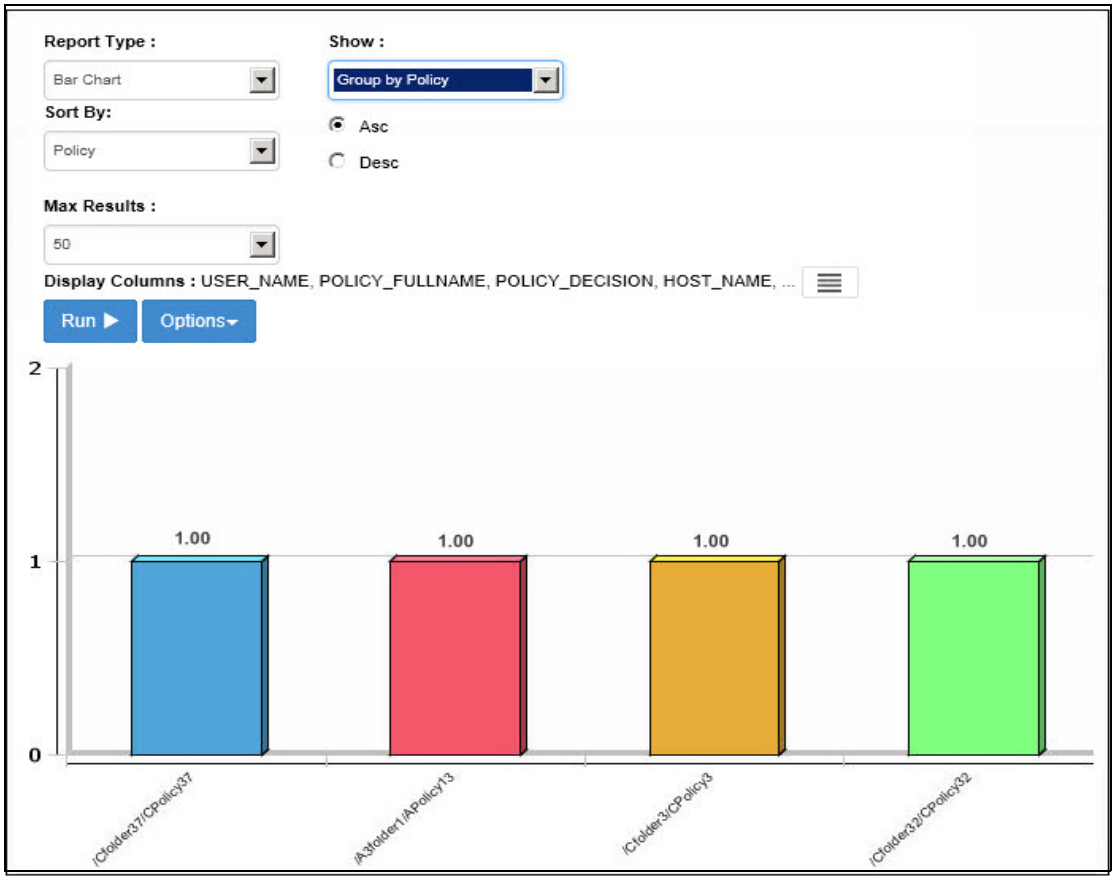


Figure 2.6-2: Group By Policy Report

This format is useful for identifying policies that are being triggered with unexpected frequency, which may be an indication that they are improperly designed and cover users, resources or actions that they should not. It can also be an indication of concentrated efforts at unauthorized data access. To look more closely into the latter possibility, it is often helpful to switch to Group by User format, to focus on who is performing the activity.

Group By User reports

Data can be grouped by user, as shown in Figure 2.6-3. The four policies are each triggered by different users, and so the graph shows four bars—each representing one user. Each is labeled with a username. In this example the bars are the same height, since each of the four users triggered a policy one time.

Figure 2.6-3: Group By User Report

All things being equal, in a live environment there will generally be little difference among users' rates of triggering policies. In cases where a big difference does arise—when one user shows a significantly disproportionate rate of triggering policies (especially Deny policies)—then the situation will likely merit closer investigation. Figure 2.6-4 shows how such a case might look in a Grouped by User Report.

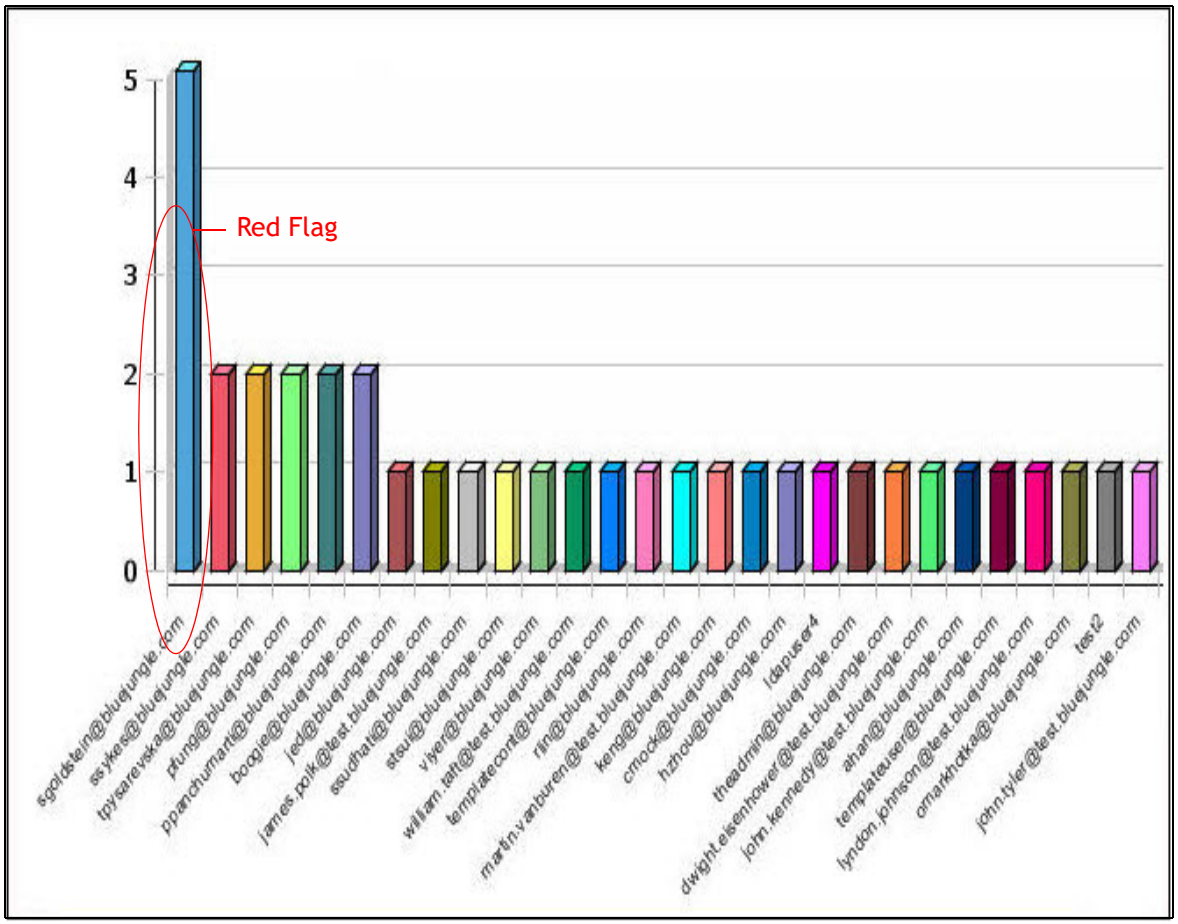


Figure 2.6-4: Group By User Report—Red Flag

Group By Resource reports

The next format type, Group by Resource, shows the extent of specified events—in this case, policies being triggered—per each individual data resource covered by the report. In our case, we know that of the four instances, two involved the same RTF file (shown in blue) and the other two, one file each (red and orange). This means three resources total, as reflected in the pie graph below.

Because policies often cover very large numbers of individual documents or other resources, this format will only be helpful when the number of incidents has already been narrowed down to a sample by various report filters, such as policies or users. This is why a pie chart format is manageable in this context. Pie charts are preferable here since in the context of resource use, the *relative* access activity regarding some single file or other resource as compared to all others, will generally be of more interest than any *absolute* number of instances of access.

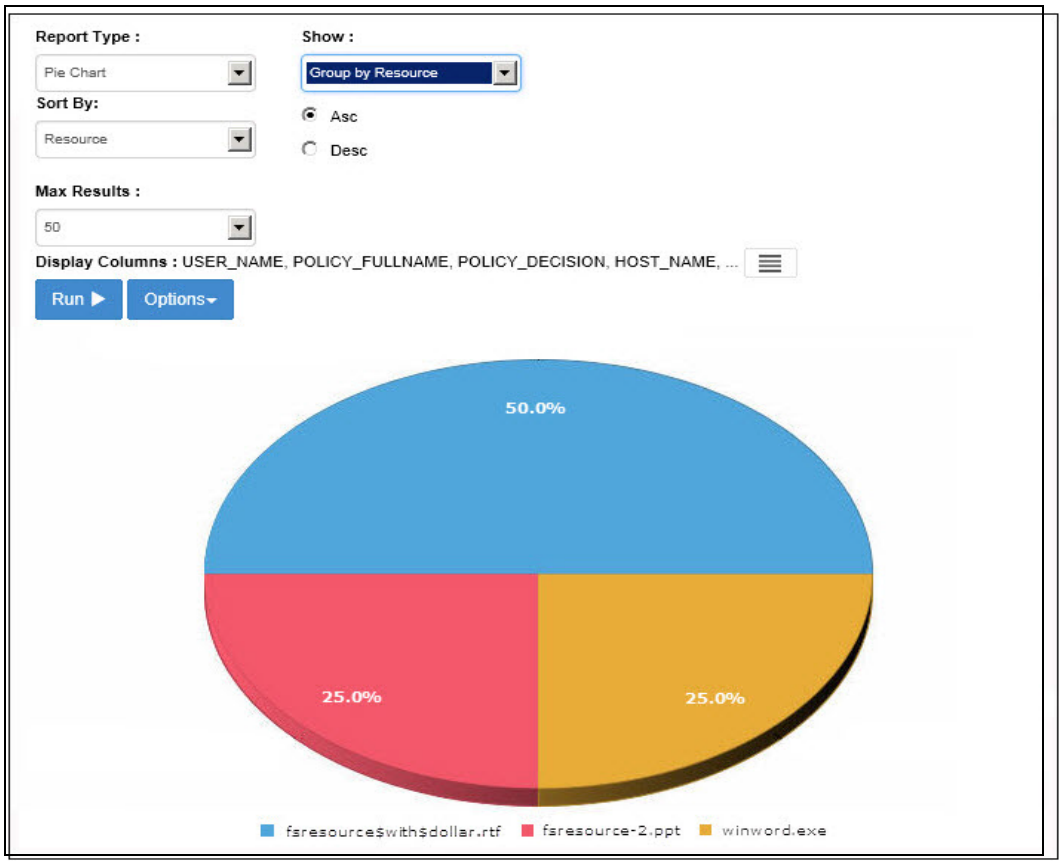


Figure 2.6-5: Group By Resource Report

Group By Time reports

When you select Show Group By Time and re-run the report all reported events are grouped by time, as shown in Figure 2.6-6. Since we know that all four events in the report occurred on the same day, this chart shows only one bar, labeled with that date. The height of the bar indicates that four incidents occurred on that date—also as expected.

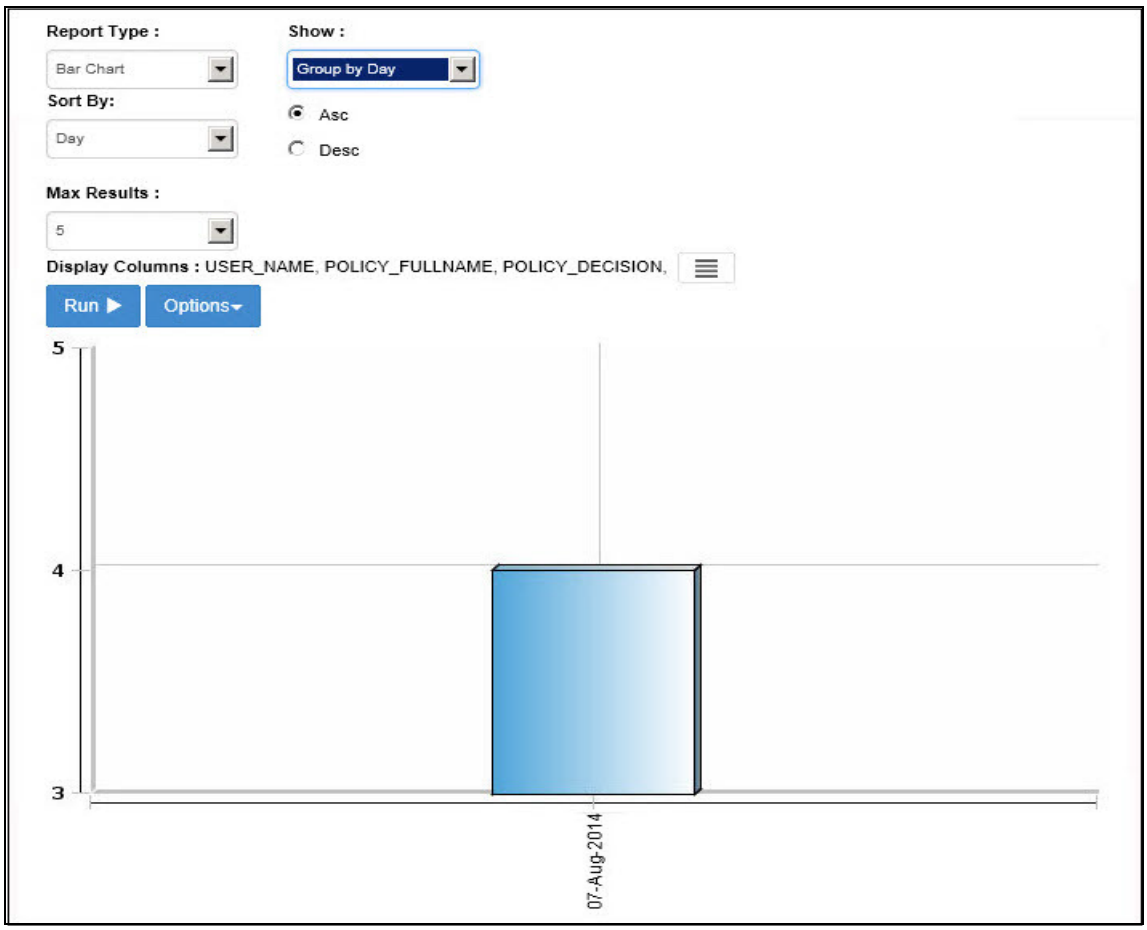


Figure 2.6-6: Group By Time Report

In a live environment, this format is particularly useful in identifying time-related enforcement spikes, which are similar to the user-related red flags. Spikes in policy enforcement at specific times such as evenings or weekends (especially with regard to Deny policies) can indicate intentional efforts at unauthorized access to sensitive resources.

2.7 Report log views

This section explains how to design custom reports, or log views, in Reporter. Reporter is the console used to monitor and report on policy compliance, gather statistics about document usage, and investigate any suspected incidents of information mishandling. Reporter is used by administrators, IT staff, managers, executives, auditors, and other authorized personnel.

Topics in this section

About report log views

Report developers can design custom reports against NextLabs policy activity data that has been written to the Activity Journal database. Several public views have been created to transform data so that queries can be run against the report schema for third-party reporting tools, such as SAP® Crystal Reports®. This section describes this public schema.

The report log views are represented in the Figure 2-7-1. Information about the columns for each view is discussed in the section View Details.

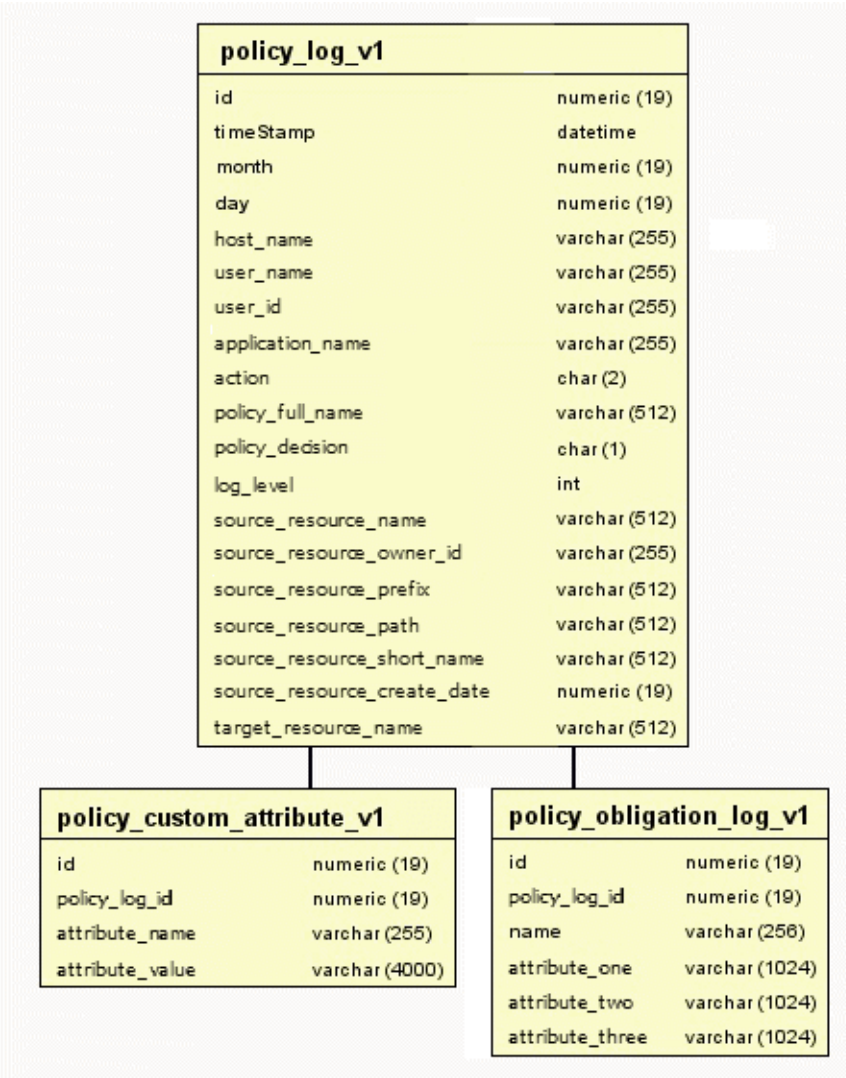


Figure 2-7-1: Overview of report log views

Policy Log Views

The main Policy Log view is **policy_log_v1**. This base view contains data related to policy enforcement, for example, the time of policy enforcement, impacted resources, actions, the policy name, the policy decision, and so on.

Two other views are associated with policy_log_v1: **policy_obligation_log_v1** and **policy_custom_attribute_v1**. The policy obligation log view contains information about custom obligations associated with a policy event. The attributes that are collected are dependent on each custom obligation (for example, for a system encryption custom obligation, this could be the path or filename of the encrypted file). The policy custom attribute log reflects resource attributes that are associated with the policy event, for example, file tags/classifications associated with resources, or attributes associated with subjects (such as citizenship for users).

View Details

This section provides more detailed information for the table column values associated with each of the Policy Log views:

- Policy Log View
- Policy Obligation Log View
- Policy Custom Attribute view

Policy Log View

Table 2.7-1 provides column value information for policy_log_v1.

Table 2.7-1: policy_log_v1

Column Name	Description	Value Type	Character
id	Record identifier	numeric	19
timestamp	Time policy enforcement occurred	datetime	
month	Timestamp for month of policy enforcement	numeric	19
day	Timestamp for day of policy enforcement	numeric	19
host_name	Fully Qualified Domain Name (FQDN) of host where policy enforcement occurred	varchar	Varies with maximum 255
user_name	User for whom policy enforcement occurred	varchar	Varies with maximum 255
user_sid	SID for user for whom policy enforcement occurred	varchar	Varies with maximum 255
application_name	Application used to access resource (full path to application)	varchar	Varies with maximum 255

Table 2.7-1: policy_log_v1 (Continued)

Column Name	Description	Value Type	Character
action	The policy action: A two-character abbreviation of a policy action. See Actions.	char	2
policy_full_name	The full name of the policy, including the path in policy tree.	varchar	Varies with maximum 512
policy_decision	The enforcement decision: • D: Deny • A: Allow.	char(1)	1
log_level	The logging level: • 1: user events, • 2: application events, • 3: system events	int	
source_resource_name	The path to the resource source for the policy. For example: file:///[mydesktop]/protected/file.doc.	varchar	Varies with maximum 512
source_resource_owner_id	The SID of the user who owns the resource.	varchar	Varies with maximum 128
source_resource_prefix	The type of resource component in the policy: file, device, portal, or sap object.	varchar	Varies with maximum 512
source_resource_path	The location where resource is stored. This differs from the full path to the resource, as in source_resource_name, as it can be a path to a directory defined in the policy. For example /[mydesktop]/protected/	varchar	Varies with maximum 512
source_resource_short_name	The filename of the source resource. For example, file.doc	varchar	Varies with maximum 512
source_resource_create_date	The date the source resource was created.	numeric	19
target_resource_name	If the resource is a policy target, as in a copy to policy, the name of the target resource. This is displayed as the full path to the file. For example: file:///[mydesktop]/protected/upload/file.doc	varchar	Varies with maximum 512

Policy Obligation Log View

Table 2.7-2 provides column value information for policy_obligation_log_v1.

Table 2.7-2: policy_obligation_log_v1

Column Name	Description	Value Type	Character Length
id	Record identifier	numeric	19

Table 2.7-2: policy_obligation_log_v1 (Continued)

Column Name	Description	Value Type	Character Length
policy_log_id	The ID of the associated policy event record in the policy_log_v1 view	numeric	19
name	Name of the custom obligation	varchar	Varies with maximum 255
attribute_one	Attribute of custom obligation (specific to obligation)	varchar	Varies with maximum 1024
attribute_two	Attribute of custom obligation (specific to obligation)	varchar	Varies with maximum 1024
attribute_three	Attribute of custom obligation (specific to obligation)	varchar	Varies with maximum 1024

Policy Custom Attribute view

Table 2.7-3 provides column value information for `policy_custom_attribute_v1`.

Table 2.7-3: policy_custom_attribute_v1

Column Name	Description	Value Type	Character Length
id	Maintained for backward compatibility with prior versions of Reporter. The value is always 0.	numeric	19
policy_log_id	The ID of the associated policy event record in the pol_log_v1 view	numeric	19
attribute_name	Name of the attribute, for example, “security_classification”	varchar	Varies with maximum of 255
attribute_value	Value of the attribute, for example, “restricted”	varchar	Varies with maximum of 4000

Actions

Table 2.7-4 provides a list of action codes that may display in the action column in `policy_log_v1`.

Table 2.7-4: Action list

Code	Action	Code	Action
Ca	Change Attributes	**Ac**	Access restricted configuration file
As	Abnormal Agent Shutdown	**Ai**	Access restricted log files
Cs	Change Security	**Ab**	Access restricted binaries
Co	Copy	**Ib**	Invalid bundle
Cp	Paste	**Br**	Bundle Received
De	Delete	**Ap**	Access agent bundle
Em	Embed	**Ex**	Export
Mo	Move	**At**	Attach
Pr	Print	**Ru**	Run
Op	Open	**Av**	Audio/Video
Ed	Edit	**Me**	Meeting
Rn	Rename	**Ps**	Presence
Se	Send Email	**Sh**	Share
Si	Send IM	**Re**	Record
Au	Start Agent	**Qu**	Ask question
Ad	Stop Agent	**Vo**	Voice
Uu	User Login	**Vi**	Video
Ud	User logout	**Jo**	Join

Reports in NextLabs Reporter

Reporter enables you to view examples of data stored in the Policy Logging views and drill down to event details, as shown in Figure 2-7-2.

The base view, policy_log_v1, provides information that populates the top-level report. When you click a specific policy enforcement event, a Log Details report opens and displays the details of that event. The two related views, policy_custom_attribute_v1 and policy_obligation_log_v1, populate the Custom Attributes and Obligation sections of the Log Details report.

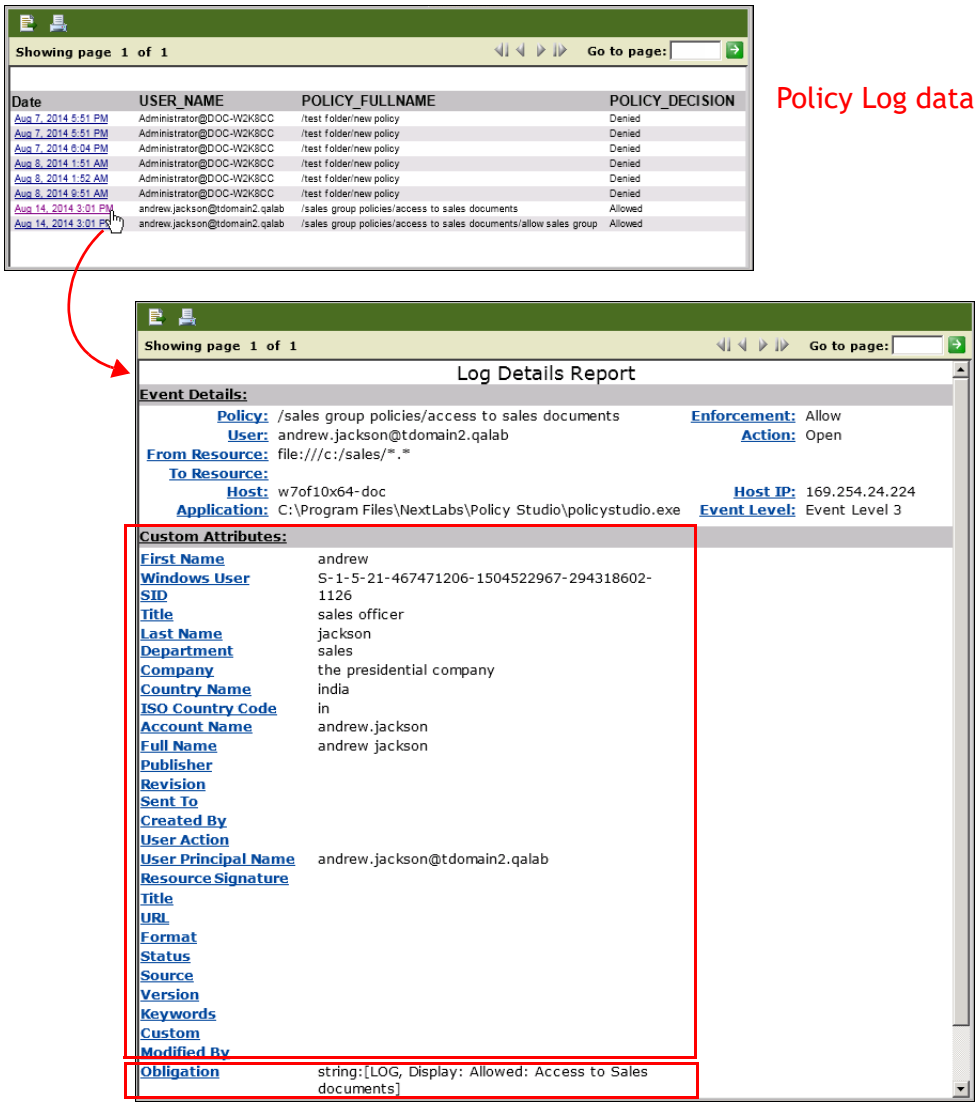


Figure 2-7-2: Policy Log Data in a NextLabs Policy Activity Report

A FAQs and troubleshooting

Question	Answer
The system automatically logged me out when I was working on a policy. Why did this happen, and will I lose my work?	For security, sessions are automatically logged out after 30 minutes of inactivity. The session timeout period cannot be changed. When you are logged in to the console and working on policies, your work is automatically saved as you go; you do not need to explicitly save your work. If you are interrupted while working on a policy, or you want to work on another task and return to authoring the policy later, you can stop and continue the authoring process as desired; your work is saved as a draft.

Glossary

This section defines the specialized terms and concepts related to NextLabs CloudAz.

A

ABAC

Attribute based access control. ABAC uses attributes, or properties, such as information about the user or the environment, in authorization requests for access to assets. Access is allowed or denied based on policies. For more information, go to: https://nccoe.nist.gov/projects/building_blocks/attribute_based_access_control.

Action components

Components used to define actions in policies. Action components serve as the verbs in policies: read, copy, print, cut and paste, attach to email, and so on. After they are defined, action components are available as the logical verbs in ACPL, the language in which policies are expressed. Action components are defined in the *Components* section of the console.

Active Control Policy Language (ACPL)

The language used internally to represent, store, and manage components and policies. Referred to as *ACPL* ("*ACK-pull*") it is also used to express advanced conditions in component and policy properties. See Using Advanced Conditions.

Activity Journal

One of the four data stores used by CloudAz. The Activity Journal stores information sent by all enforcers about how subjects access and use information, and also on policy enforcement. When you use the Reporter console to generate reports, you are querying data from the Activity Journal. The Activity Journal can be maintained either on the internal database supplied with CloudAz, or on an external Oracle, PostgreSQL, or Microsoft SQL database.

alerts

Notifications that are sent to administrators when specified policy enforcement activities occur. Together with monitors, alerts can provide continuous coverage of key policy enforcements in an enterprise, as well as a notification system that warns administrators when action is required.

auditor

A logical component of Policy Enforcer software, responsible for capturing document access and use by policy subjects, as well as all policy enforcement events, and writing them to the *Activity Journal* database.

B

bundle

See *policy bundle*.

C

cancel

One of the actions you can take to change the deployment status of policies and components. You can cancel an object if it has been submitted and scheduled for deployment, but not if it has already been deployed. Deployed objects must be *deactivated* before they can be canceled.

components

The building blocks of policies. Components represent classes or categories of entities in the physical network environment.

Control Center

The enterprise-wide platform where policies are constructed, stored, and managed, and where audit data is gathered and managed, also known as CloudAz. CloudAz comprises three major pieces—the *management server*, the *Report Server*, and *ICENet server*—plus several data stores, which are generally stored in a dedicated, external database.

D

deactivate

The act of changing a policy or component from a deployed state to an inactive state. Deactivated objects can be reactivated later or deleted as needed. Objects that have been deployed must be deactivated before they can be deleted.

Delegated Administration

A method of managing user access to the CloudAz consoles. Delegated Administration enables administrators to manage user access to the CloudAz consoles as well as policies, and policy objects, by creating rules based on user attributes and conditions. Using rules and attributes mirrors the way policies are used to control access to resources and eliminates the need to create and apply roles to users or groups of users.

delete

See *deactivate*.

deployed

One of the possible states of a policy or component. When an object is deployed, it is distributed to all relevant enforcers in the system. It then is used to govern the way components access and use resources.

E

effect

The enforcement action to be taken when a policy is enforced. Effects include Deny and Allow.

H

heartbeats

Regular messages sent along the communications channel between Cloud PDPs and the CloudAz server. Cloud PDPssend regular heartbeats to the CloudAz server to indicate that they are running normally. When a heartbeat occurs, the Cloud PDP receives any pending policy deployments or configuration updates from CloudAz.

Heartbeat plug-ins use this communication channel between CloudAz and Cloud PDPsto send and receive other types of data. For example, you can build a heartbeat plug-in to send external authorization data, such as licenses, to all Cloud PDPsor to send installation or upgrade information about each Cloud PDP to store in CloudAz.

I

ICENet

A protocol used for communication between policy enforcers and the CloudAz *Control Center*. All such communication passes through the *ICENet server* on the set of software servers and other modules that constitute the heart of the CloudAz platform. The main components of the CloudAz are the Policy Management Server and Policy Master data store, the Management Server and Information Network Directory, the Intelligence Server and Activity Journal, one or more ICENet Servers, and the Reporter console. Each enforcer is equipped with an *ICENet client*.

ICENet client

The component that handles remote communication with the *ICENet server*.

ICENet server

The CloudAz component that manages communication between all policy enforcers and the set of software servers and other modules that constitute the heart of the CloudAz platform. The main components of the CloudAz are the Policy Management Server and Policy Master data store, the Management Server and Information Network Directory, the Intelligence Server and Activity Journal, one or more ICENet servers, and the Reporter console server

M

management database

One of the three databases used by CloudAz. The management database stores data about authorized users.

management server

The CloudAz component that centralizes management of all system components.

monitors

Predefined criteria that identify the policy enforcement activities to be tracked. Together with alerts, monitors can provide continuous coverage of key policy enforcements in an enterprise, as well as a notification system that warns administrators when action is required.

O

object components

One of the two general categories of components; the other is *Activity Journal*. Object components provide the subjects and objects in policies.

obligations

Instructions sent by the PDP (Policy Decision Point) to the PEP (Policy Enforcement Point) along with the authorization decision. For example, an obligation might instruct the PEP to notify the user why they are not allowed to perform a certain operation, and the message itself might be included in the obligation. In other cases, obligations might instruct the PEP to record the allowed user action in an audit database; encrypt a value before allowing the user to store that value; or inject a security filter in a SQL query to restrict the rows being returned, before allowing the user to run the query against a given database table.The actions CloudAztakes, in addition to policy enforcement actions, such as allowing or denying user actions, when enforcing policies. There are three types of obligations: make a log entry in the *Activity Journal*, display a notification to the subject, and send an email notification to a specified recipient. Policies can include one or more of each of these obligation types. Log entries are required for all *On Deny* enforcements.

P

PAP

See *Policy Administration Point*.

PDP

See *Policy Decision Point*.

PEP

See *Policy Enforcement Point*.

PIP

See *Policy Information Point*.

policy

A rule designed to control information use in an organization. Policies consist of *components* combined with logical operators, variables such as *obligations*, and time-based contexts.

Policy Administration Point

The CloudAz server.

- **Policy management service**: The service that provides the interface used to author and manage policies.
- **ICENet**: A server component that communicates with the Cloud PDP to send policies and receive user activity records.
- **Reports, Monitoring and Alerts**: The interface used to view and track user activity.
- **Delegated Administration:** The interface used to control user access to the CloudAz console.
- **Database repositories**: Used to store policies and audit records.

policy bundle

The set of policies and required component definitions that are defined and active for each enforcer running in the system. Each enforcer has only one bundle stored locally at any one time, and it contains all the information it requires for all policies currently deployed to it. Every heartbeat interval, each enforcer contacts the CloudAz inquiring if there are any new or updated policies that should be deployed to that enforcer; if there are, the CloudAz sends a new bundle that overwrites the one currently deployed there.

policy component

See *components*.

Policy Decision Point (PDP)

The entity that evaluates applicable policies and computes authorization decisions. CloudAz comes with a preconfigured set of PDPs (Policy Decision Points), which supports standards-based interaction REST APIs to make authorization requests. The PDPs evaluate deployed authorization policies at runtime against incoming requests, and return authorization decisions to PEPs (Policy Enforcement Points).

Policy Enforcement Point (PEP)

The software component that performs access control by making authorization requests to PDPs (Policy Decision Points) and enforcing decisions.

Policy Information Point (PIP)

The component that is the source of user, resource, and environment attributes. A PEP (Policy Enforcement Point) or PDP (Policy Decision Point) interacts with the PIP to obtain attributes. In an enterprise deployment, a typical PIP would be an LDAP server, HR database, or other data repository with license keys.

policy modeling

The process of defining the metadata to represent actors (Subjects) requesting authorization for a given set of objects (Resources).

policy object

A generic term sometimes used to refer to policies and policy components.

R

Reporter console

One of the user interfaces in CloudAz. The Reporter console is a web-based application used to create reports as well as monitor and track user activity. It is the front end of the *Report Server*.

report query

The group of filter settings available in the Reporter console that control how report content is extracted from the *Activity Journal*.

reports

The result of a query run on information stored in the *Activity Journal*, showing details related to information access and use and policy enforcement in the network. Reports can be displayed in detail (grid) format or as bar charts.

Report Server

A major component of CloudAz that is responsible for storing, managing, and serving data on end-user activity and policy enforcement. This information is stored in the *Activity Journal* database. The Report Server is the back end of the Reporter console.

report settings

The group of settings available in the Reporter console that control how reports are formatted.

resource components

Components that can be included in policy definitions. Resources, such as documents or servers, are the grammatical object of actions in the ACPL language.

resource types

The policy model templates that include information about attributes, actions, and obligations available to components and policies.

S

shared reports

Reports that are available to multiple users. When reports are saved, they can be marked as shared. Shared reports appear on a separate tab in the Reporter console. See Saving reports.

states

The current status of a policy or a component. There are seven possible states: Editing, Submitted for Deployment, Deployed, Deactivated, and Deleted. An object's state determines the actions that can be performed on it.

system details

The Dashboard widget that displays details about the configuration of the system. It shows how many Cloud PDPs are currently active, and how many Audit records are currently stored. It also provides subscription details.

Index

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X