Guidewire ClaimCenter™ for Guidewire Cloud

Cloud API Authentication Guide

Release: 2021.11

GUIDEWIRE

Product Name: Guidewire ClaimCenter for Guidewire Cloud
Product Release: 2021.11
Document Name: Cloud API Authentication Guide
Document Revision: 19-March-2023

Contents

Support 7

Part 1
Choosing an authentication flow

1 **Overview of authentication** 11
Types of callers 11
Authentication architecture 12
Types of access 13
Authentication methods 13
Authentication failure error messages 14
List of developer tasks 15
2 **Selecting an authentication flow** 17
Auth flows to choose from 17
Detailed discussion of issues to consider 18
Which OAuth flow must the caller application use? 18
Which user is attached to the session? 19
Where do authorization values come from? 20
Who enforces resource access? 21
What values are used as resource access IDs? 22
Summary of the issues to consider 22
Additional auth flows 23

Part 2
Authentication flows in detail

3 **Basic authentication** 27
Overview of basic authentication 27
Credentials 27
Authorization 27
Request headers 28
Example flow for basic authentication 29
Implementation checklist for basic authentication 30
Sending authenticated calls with basic authentication 30
Send a Postman call with basic authentication 31
4 **OAuth2 authorization code flow: Internal users** 33
Overview of authentication for internal users 33
Credentials 33
Authorization 33
JWTs for internal users 34
Logging 35
Example flow for internal users 35
Implementation checklist for internal users 37
Sending authenticated calls for internal users 37
5 **OAuth2 authorization code flow: External users** 39

Overview of authentication for external users....39
Credentials....39
Authorization....39
JWTs for external users....41
Logging....42
Example flow for external users....42
Implementation checklist for external users....44
Sending authenticated calls for external users....44
6 OAuth2 client credential flow: Standalone services....47
Authentication options for services....47
Overview of authentication for standalone services....49
Credentials....49
Authorization....49
JWTs for standalone services....50
Logging....51
Example flow for standalone services....51
Implementation checklist for standalone services....53
Sending authenticated calls for standalone services....53
7 OAuth2 client credential flow: Services with user context....55
Authentication options for services....55
Overview of authentication for services with user context....57
Credentials....57
Authorization....57
JWTs for services with user context....59
Logging....60
Example flow for services with user context....60
Implementation checklist for services with user context....65
Sending authenticated calls for services with user context....66
8 OAuth2 client credential flow: Services with service account mapping....67
Authentication options for services....67
Overview of authentication for services with service account mapping....69
Credentials....69
Authorization....69
JWTs for services with service account mapping....70
Mapping services to service accounts....71
Logging....72
Example flow for services with service account mapping....72
Implementation checklist for services with service account mapping....74
Sending authenticated calls for services with service account mapping....74
9 Unauthenticated callers....77
Overview of authentication for unauthenticated callers....77
Credentials....77
Authorization....77
JWTs for unauthenticated callers....78
Logging....78
Example flow for unauthenticated callers....78
Implementation checklist for unauthenticated callers....80

Part 3
Implementing authentication

10 Enabling bearer token authentication ... 83
Registering ClaimCenter with Guidewire Hub ... 83
Enabling asymmetric encryption ... 83
Enable asymmetric encryption ... 83
Specifying deployment information ... 84
Configuring the IdP ... 84
Configure the IdP for bearer token authentication ... 84
Registering the caller application with Guidewire Hub ... 85
Register an application with Guidewire Hub ... 85
11 Endpoint access ... 87
API role files ... 87
API role names ... 88
API role endpoints ... 88
API role accessible fields ... 89
API role example ... 90
Assigning API roles to callers ... 90
Assigning API roles to internal users ... 90
Assigning API roles to external users and services ... 91
Assigning API roles to other types of callers ... 91
Reserved roles ... 92
Designing API role files ... 92
Configuring API roles ... 92
Create an API role file ... 92
Modify an API role file ... 93
API roles and localization ... 93
12 Resource access ... 95
Resource access strategies ... 95
Resource access files ... 97
Permissions and filters ... 98
Resource permissions ... 98
Resource filters ... 99
Configuring resource access ... 99
13 Proxy user access ... 101
Proxy users ... 101
When is proxy user information used? ... 103
Configuring proxy users ... 103
Create a new proxy user ... 104
14 ContactManager authentication ... 105
Supported caller types ... 105
Resource access for ContactManager ... 105
Tag-based access to contacts ... 105

Support

For assistance, visit the Guidewire Community.

Guidewire customers

https://community.guidewire.com

Guidewire partners

https://partner.guidewire.com

part 1

Choosing an authentication flow

Endpoints withing Cloud API must control access to the data and actions within ClaimCenter. When a caller tries to access data or execute an action, the caller must be authenticated and authorized. *Authentication* is the process of verifying that the caller is who they claim to be. *Authorization* is the process of determining what operations and data the caller is allowed to access. These two process are often referred to collectively as "auth".

The following topics provides an overview of the different aspects an insurer must consider when planning an authentication approach. This includes:

- The different types of callers that Cloud API supports
- The different applications involved with Cloud API authentication
- The types of access enforced by Cloud API
- The supported authentication methods

This concludes with a topic that can help insurers determine which authentication flows are most appropriate for a given caller application.

chapter 1

Overview of authentication

The system APIs must control access to the data and actions within ClaimCenter. When a caller tries to access data or execute an action, the caller must be authenticated and authorized. *Authentication* is the process of verifying that the caller is who they claim to be. *Authorization* is the process of determining what operations and data the caller is allowed to access. These two process are often referred to collectively as "auth".

This topic provides an overview of how authentication and authorization are managed for the system APIs.

Types of callers

Within the context of system API authentication, a *caller* is a user or service who triggers a system API call from a caller application.

There are several different types of callers. This documentation uses the following terms to identify them:

- **Internal user** - This is a person who is listed as a user in the ClaimCenter operational database. For example, Andy Applegate, a ClaimCenter adjuster, is an internal user.
  - Note that internal users can use caller applications and trigger system API calls from those applications. For example, suppose there is a loss documentation portal that contains pictures of a damaged auto taken by a third-party field agent. An adjuster reviews and selects pictures to be saved to ClaimCenter. This action triggers a system API call by an internal user from a caller application.
- **External user** - This is a person who is known to the insurer but who is not listed as a user in the ClaimCenter operational database. For ClaimCenter, there are two typical types of external users:
  - **Policyholders** - Users who want to interact with information about claims on their policies. For example, Ray Newton, who is a policyholder and wants to check on the status of a claim filed against his personal auto policy.
  - **Service providers** - Users who want to interact with service requests. For example, Mike's Auto Detailing Shop, who wants to accept a request to repair Ray Newton's vehicle.
- **Service** - This is a service, also referred to as a *service-to-service application*. For example, a service that periodically creates new claims based on FNOL information entered into an external system. There are several ways in which a service can make a call:
  - As a **standalone service**, in which the service executes the call as itself. It does not execute the call on behalf of a specific person or through a ClaimCenter user account.
  - As a **service with user context**, in which the service presents information about itself and about a specific user. The call is able to do only the things that both the service by itself could do and the user by itself could do.
  - As a **service with service account mapping**, in which the service is mapped to an account in the ClaimCenter database and has access as determined by that account.

  - Do not confuse service providers and services. A *service provider* is a person or business who is an external user that provides assistance to a claimant. A service provider can also be referred to as a vendor. However, a *service* is a process or application that can execute action without direct human interaction.
- **Unauthenticated caller** - This is a user or service who provides no authentication information. Unauthenticated callers can access only metadata endpoints. Unauthenticated callers are typically callers who need information about the system APIs.

Within the context of authentication and authorization, this documentation uses the following terms in the following way:

- *User* is used exclusively for callers that are people.
- *Service* is used exclusively for callers that are not people and that take action without direct action from a person.
- *Service account* is used to refer to an account in the ClaimCenter database that is used exclusively by a service and that defines access for that service.
- *Caller* is used to collectively refer to users and services.

Authentication architecture

The authentication architecture for system APIs consists of:

- The InsuranceSuite application (such as ClaimCenter)
- Guidewire Identity Federation Hub
- The insurer's identity provider (IdP)
- A set of one or more caller applications

Note that some parts of the architecture are relevant for all system API calls, regardless of the type of caller. Other parts of the architecture are relevant only for certain types of callers.

Guidewire Hub

Guidewire Identity Federation Hub (Guidewire Hub) is the trusted auth server for all Guidewire cloud applications, including caller applications that insurers create to access Guidewire cloud resources. Guidewire Hub uses OAuth 2.0 and SAML for identity management services.

The primary responsibilities of Guidewire Hub are:

- For internal users and external users:
  - To receive authentication requests from InsuranceSuite applications and caller applications
  - To federate those authentication requests to the correct IdP
  - To construct JWTs that verify users and identify their authorization
- For services:
  - To authenticate services
  - To construct JWTs that verify services and identify their authorization

The insurer's identity provider

An *identity provider* (IdP) is an application or service that creates, maintains, and manages identity information for internal and external users. Every insurer using Guidewire cloud applications must provide an identity provider (IdP).

The primary responsibilities of the IdP are:

- For internal users and external users:
  - To maintain user names and passwords
  - To maintain information that identifies each user's authorization
  - To authenticate users and provide their authorization information when a request is received from Guidewire Hub

The IdP does not play a role in service authentication or authorization.

The caller applications

Every caller application that uses system APIs must provide authentication information with every API call (except for unauthenticated calls).

From an authentication perspective, the primary responsibilities of each caller application are:

- For internal users and external users:
  - To send authentication requests to Guidewire Hub (which will then federate those requests to the appropriate IdP)
- For services:
  - To send authentication requests to Guidewire Hub (which are executed by Guidewire Hub without any involvement of the IdP)
- For all callers:
  - To temporarily store JWTs created by Guidewire Hub so that they can be included in system API calls made for the associated callers

The InsuranceSuite system APIs

From an authentication perspective, the primary responsibilities of the system APIs are:

- For authenticated callers:
  - Verify that each API call includes valid authentication
  - Limit the access of each API call to only those endpoints, operations, fields, and specific resources that the user is authorized to use
- For unauthenticated callers:
  - Limit the access of each API call to the appropriate endpoints, operations, fields, and resources
    - Typically, this access is limited to API metadata only

Types of access

Authorization is the process of determining what operations and data the caller is allowed to access. The system APIs enforce authorization using the following types of access:

- *Endpoint access* defines the aspects of an endpoint's behaviors that are available to a caller. This includes:
  - What endpoints are available to the caller?
  - What operations can a caller call on the available endpoint?
  - What fields can the caller specify in a request payload or get in a response payload?
- *Resource access* defines, for a given type of resource, which instances of that resource type the caller can access. For example, for a given caller, endpoint access might grant access to a GET `/policies` endpoint. But this does not necessarily mean the caller can access every policy in the system. Resource access can limit which specific policies that caller can view.
- A *proxy user* is an internal user that is assigned to an external user or service when an external user or service triggers an API call. Whenever ClaimCenter logic must verify that the caller has a given domain-level system permission (such as permission to own an activity) or authority limit, the proxy user is checked. This is referred to as *proxy user access*.

Each type of access does not necessarily apply to every type of caller.

- All types of users are bound by endpoint access.
- In the base configuration, only users are bound by resource access. Services are not.
- Only external users and services are bound by proxy user access. Internal users are not.

Authentication methods

The system APIs support two authentication methods. The methods differ based on how authentication information is sent from the caller application to ClaimCenter.

Basic authentication

Basic authentication is an authentication method in which only the user's user name and password are provided, and they are provided in the request header.

- Internal users (and only internal users) can use basic authentication.
- With basic authentication, the authentication and authorization information is retrieved from the operational database using information in the request header.

Guidewire recommends using basic authentication only over HTTPS (SSL).

Bearer token authentication

Bearer token authentication is an authentication method in which the authentication information is stored in a JSON Web Token (JWT, pronounced like "jot"). The phrase "bearer authentication" can be understood as "give access to the bearer of this token".

- Every type of caller can use bearer token authentication.
- With bearer token authentication, the JWT contains both authentication information and authorization information.

JWTs contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

Similar to basic authentication, Guidewire recommends using bearer token authentication only over HTTPS (SSL).

Authentication failure error messages

Endpoints that return elements

For endpoints that return elements, when a given resource does not exist, Cloud API throws a `NotFoundException` with a user message similar to "No resource was found at path...". When a given resource does exist, but the user lacks sufficient resource access, Cloud API throws the same exception with the same user message. This approach is considered to be more secure as it prevents malicious callers from being able to verify the existence of data that they are not authorized to access.

For example, suppose a user executes GET `/activities/xc:20`. Also, suppose activity xc:20 exists but the user lacks sufficient resource access, the following error is returned:

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path /activities/xc:20"
```

Endpoints that return collections

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient resource access, Cloud API omits it from the results. This approach is considered to be more secure as it prevents malicious callers from being able to verify the existence of data that they are not authorized to access.

For example, suppose a user executes GET `/activities`. Also, suppose that there are three activities in the database: xc:10, xc:20, and xc:30. The user has sufficient resource access to view xc:10 and xc:30, but not xc:20. The call returns the following:

```
{
  "count": 2
  "data": [
    {
      "attributes": {
        "id": "xc:10",
        ... },
      ...
    },
    {
```

```
      "attributes": {
        "id": "xc:30",
        ... },
      ...
    }
  "links": { ... }
}
```

List of developer tasks

There are a number of tasks an insurer must execute to enable and use authentication for each type of caller.

Configuring authentication

During implementation, for a given type of caller, an insurer may need to:

- Enable asymmetric encryption
- Provide deployment information
- Register the caller application with Guidewire Hub
- Configure the IdP
- Configure service account mapping
- Configure endpoint access
- Configure resource access
- Configure proxy user access

Making API calls

In production, for a given type of caller, the caller may need to:

- Request a code from Guidewire Hub
- Request a JWT from Guidewire Hub
- Send an API call to ClaimCenter with a JWT

chapter 2

Selecting an authentication flow

Within the context of Cloud API, an *auth flow* is a flow of authentication and authorization information for a particular type of caller. Cloud API supports multiple auth flows. This topic identifies the issues to consider when choosing an auth flow for a particular caller application.

The most important issues to consider are as follows:

- What OAuth flow must the caller application use?
- What user is attached to the session?
- Where are authorization values stored?
- Who enforces resource access?
- What values are used as resource access IDs?

This topic assumes you are familiar with the Cloud API authentication architecture and the meaning of the terms endpoint access, resource access, and proxy user access. For more information, see "Overview of authentication" on page 11.

Auth flows to choose from

Cloud API supports the following auth flows for bearer token authentication.

Auth flow	Definition of who the caller is
Internal user	A person who is known to the insurer and listed as a user in the ClaimCenter database, such as a claims adjuster or underwriter.
External user	A person who is known to the insurer, but who is not listed as a user in the ClaimCenter database, such as a policy holder.
Standalone service	A service that executes calls as itself.
Service with internal user context	A service that executes calls on behalf of internal users.
Service with external user context	A service that executes calls on behalf of external users.
Service with service account mapping	A service that is mapped to an internal service account and whose access is determined by the settings for that service account.

Detailed discussion of issues to consider

When determining the best bearer token auth flow for a given caller application, Guidewire recommends you consider the issues discussed in the following topics.

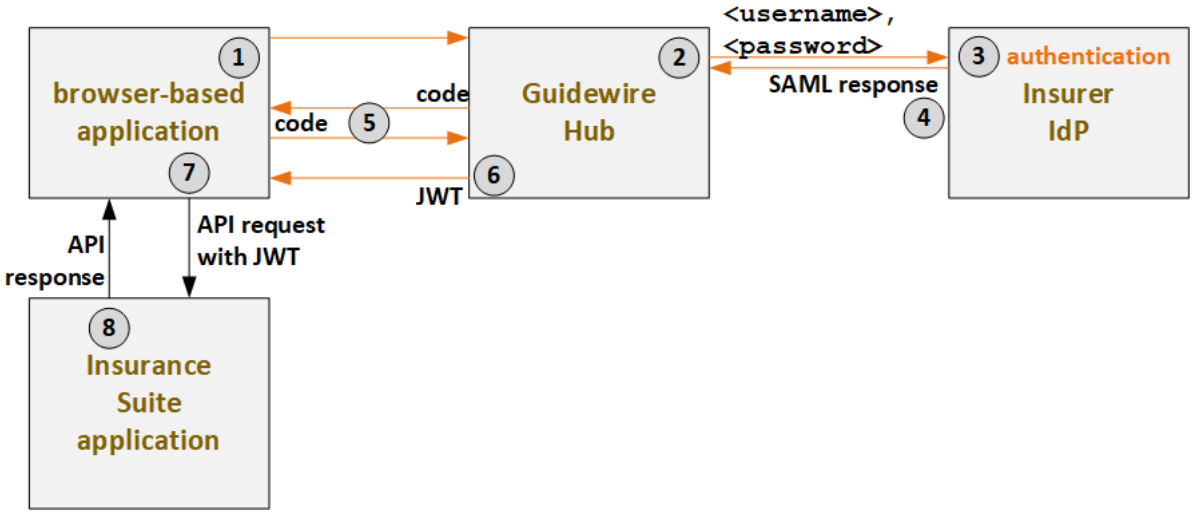
For a summary of the issues, and the behaviors for each auth flow, see “Summary of the issues to consider” on page 22.

Which OAuth flow must the caller application use?

Cloud API supports two OAuth flows: authorization code flow and client credential flow.

Authorization code flow

Authorization code flow is designed for browser-based applications that typically have a user interface and that users interact with.



Within authorization code flow:

1. The caller application requests a JWT from Guidewire Hub.
2. Guidewire Hub acquires the user's user name and password from the user. It sends this information to the appropriate IdP.
3. The IdP authenticates the user.
4. The IdP provides a SAML response with the information that defines the user's authorization. This could include endpoint access role names and resource access IDs.
5. Guidewire Hub sends a code to the caller application. The caller application uses this code to request a JWT.
6. Guidewire Hub sends the JWT to the caller application.

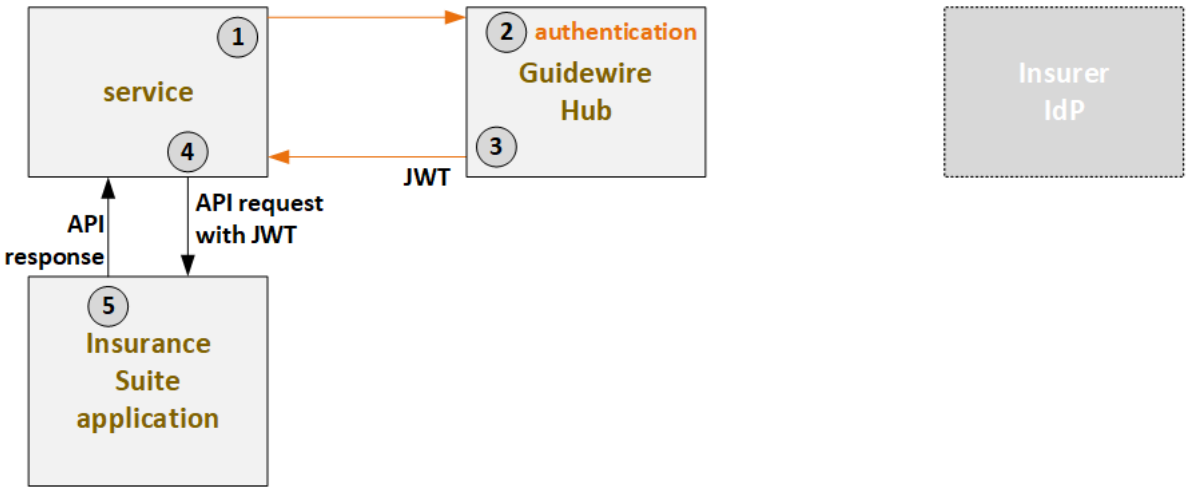
Once the caller application has the JWT, it can send the API request to ClaimCenter (7), which then processes the request and sends the reply (8).

Authorization code flow can be used with the following Cloud API auth flows:

- Internal user
- External user

Client credential flow

Client credential flow is designed for services that typically do not have a user interface and that take action without any synchronous user input.



Within client credential flow:

1. The caller requests a JWT from Guidewire Hub.
2. Guidewire Hub authenticates the caller.
3. Guidewire Hub sends the JWT to the caller application.

Once the caller application has the JWT, it can send the API request to ClaimCenter (4), which then processes the request and sends the reply (5).

Client credential code flow can be used with the following Cloud API auth flows:

- Standalone service
- Service with internal user context
- Service with external user context
- Service with service account mapping

Summary of behaviors

The following table summarizes these behaviors.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
OAuth flow	Authorization code flow	Authorization code flow	Client credential flow	Client credential flow	Client credential flow	Client credential flow

For a summary of all the issues to consider in a single table, see “Summary of the issues to consider” on page 22.

Which user is attached to the session?

Every session must have a user attached to it. This user is used in the following way:

- If the call creates or modifies data, the user name is recorded as the `CreateUser` or `UpdateUser`.
- If the call creates a history event, the user is attached to the event.
- If the call assigns an object, the user name is used in "assigned by *user*" information.

Also, once Cloud API has confirmed there is sufficient authorization to use a given endpoint and view a given resource:

- If the call also triggers an authority limit check, the user's authority profiles are checked.
- If the call also triggers a domain-level permissions check, the user's permissions are checked.

Every caller can have their own user account

For some auth flows, every caller can have their own user account. In these situations, session users are assigned on a per-caller basis. These behaviors can also be controlled separately for each caller. The following Cloud API auth flows support this:

- Internal user
- Service with internal user context
- Service with service account mapping

Multiple callers share a single proxy user account

For other auth flows, multiple callers share a single proxy user account. This occurs in situations where individual callers are not listed in the ClaimCenter database, and therefore session users cannot vary from caller to caller. For these situations, a single proxy user is assigned to an entire type of caller (external user or service). The following Cloud API auth flows support this:

- External user
- Standalone service
- Service with external user context

Summary of behaviors

The following table summarizes these behaviors.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
Can each call have its own user attached to the session?	Yes	No (a single "external proxy user" is used for all relevant calls)	No (a single "service proxy user" is used for all relevant calls)	Yes	No (a single "service proxy user" is used for all relevant calls)	Yes

For a summary of all the issues to consider in a single table, see “Summary of the issues to consider” on page 22.

Where do authorization values come from?

For every caller, you need to store a value that determines the caller's endpoint access. This value determines which endpoints and operations the caller can use. For some callers, you must also store values that determine the caller's resource access. These values determine which specific instances of a given resource the caller can view or edit. These are collectively referred to here as *authorization values*.

The IdP

For some auth flows, authorization values must come from the IdP, or otherwise accessible to the IdP so that they can be included in the IdP's SAML response. The following Cloud API auth flows support this:

- Internal user
- External user

The caller application itself

For other auth flows, authorization values must come from the caller application itself, of otherwise accessible to it. The following Cloud API auth flows support this:

- Standalone service
- Service with internal user context
- Service with external user context

The Guidewire configuration

For the service with service account mapping auth flow, the caller application provides a client ID. This ID is mapped to a service account, and this service account is used to determine the authorization values. For this auth flow, the authorization values come from the mapping information, which is stored in the Guidewire configuration itself.

Summary of behaviors

The following table summarizes these behaviors.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
Where do authorization values come from?	The IdP	The IdP	The service itself (endpoint access values only; resource access IDs are not applicable)	The service itself	The service itself	The Guidewire configuration

For a summary of all the issues to consider in a single table, see "Summary of the issues to consider" on page 22.

Who enforces resource access?

Cloud API

For some auth flows, resource access is enforced by Cloud API. For these auth flows, the call either includes a resource access ID, or includes information that can be mapped to a resource access ID. This ID determines which specific instances of a resource the caller can access. The following Cloud API auth flows support this:

- Internal user
- External user
- Service with internal user context
- Service with external user context
- Service with service account mapping

The caller application itself

For other auth flows, Cloud API provides unrestricted resource access. This is done under the assumption that resource access will be enforced by the caller application itself. In this case, calls do not include resource access IDs. The caller is given access to any specific resource, provided they have sufficient endpoint access. The following Cloud API auth flows support this:

- Standalone service

Summary of behaviors

The following table summarizes these behaviors.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
Does Cloud API enforce resource access?	Yes	Yes	No (The service is expected to enforce it.)	Yes	Yes	Yes

For a summary of all the issues to consider in a single table, see "Summary of the issues to consider" on page 22.

What values are used as resource access IDs?

User names

For some auth flows, resource access IDs are user names. For these auth flows, every call must present a user name or service account name. Resource access is determined by this name. The following Cloud API auth flows support this:

- Internal user
- Service with internal user context

Business IDs

For other auth flows, resource access IDs are business IDs. These IDs represent what the caller owns, such as policy numbers (for ClaimCenter policy holders), account numbers (for PolicyCenter account holders), or address book unique identifiers (for vendors providing services for ClaimCenter claims). For these auth flows, every call must present one or more business IDs. Resource access is determined by what the caller owns. The following Cloud API auth flows support this:

- External user
- Service with external user context

No resource IDs

There is two auth flows that do not use resource IDs.

For standalone services, resource access is enforced by the service itself, and not by Cloud API. Therefore, there is no need to provide resource access IDs.

For services with service account mapping, the service is mapped to a service account. Information in the service account is used to determine resource access, but there are no resource IDs passed within the auth flow.

Summary of behaviors

The following table summarizes these behaviors.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
What are the resource access IDs?	user names	business data (such as policy numbers and account numbers)	not applicable	user names	business data (such as policy numbers and account numbers)	not applicable

For a summary of all the issues to consider in a single table, see “Summary of the issues to consider” on page 22.

Summary of the issues to consider

The following table summarizes the issues to consider and the behavior of each auth flow regarding that issue. The last row of the table contains links to view more detailed information about the relevant auth flow.

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
OAuth flow	Authorization code flow	Authorization code flow	Client credential flow	Client credential flow	Client credential flow	Client credential flow
Can each call have its own user	Yes	No	No	Yes	No	Yes

	Internal User	External User	Standalone service	Service with Internal User Context	Service with External User Context	Service with Service Account Mapping
attached to the session?		(a single "external proxy user" is used for all relevant calls)	(a single "service proxy user" is used for all relevant calls)		(a single "service proxy user" is used for all relevant calls)	
Where do authorization values come from?	The IdP	The IdP	The service itself (endpoint access values only; resource access IDs are not applicable)	The service itself	The service itself	The Guidewire configuration
Does Cloud API enforce resource access?	Yes	Yes	No (The service is expected to enforce it.)	Yes	Yes	Yes
What are the resource access IDs?	user names	business data (such as policy numbers and account numbers)	not applicable	user names	business data (such as policy numbers and account numbers)	not applicable
For more information on this auth flow	"OAuth2 authorization code flow: Internal users" on page 33	"OAuth2 authorization code flow: External users" on page 39	"OAuth2 client credential flow: Standalone services" on page 47	"OAuth2 client credential flow: Services with user context" on page 55	"OAuth2 client credential flow: Services with user context" on page 55	"OAuth2 client credential flow: Services with service account mapping" on page 67

Additional auth flows

Cloud API supports additional auth flows that do not make use of bearer token authentication:

Auth flow	Definition
Basic authentication	The caller is a person who is known to the insurer and listed as a user in the ClaimCenter database, such as a claims adjuster or underwriter. The call uses basic authentication (where auth information is passed in the request header) and not bearer token authentication (where auth information is passed in a JWT (JSON Web Token). Basic authentication can be an appropriate option for development environments, as it bypasses the new to have a working integration with Guidewire Hub or, when relevant, an IdP.
Unauthenticated user	The caller is a person or service who presents no authentication information. Typically, this type of caller has access to metadata only. Unauthenticated user access can be appropriate for internal services that only need access to metadata, such as a service that is querying for the definition of an API.

The following table summarizes the issues to consider. It identifies the options for each issue, and which auth flow supports each option. The final row of the table provides a link which you can follow to get more detailed information about that auth flow.

	Basic Auth	Anonymous User (PolicyCenter only)	Unauthenticated User
OAuth flow	not applicable	not applicable as an anonymous user (Anonymous users who bind policies becomes external users, and from then on they use authorization code flow.)	not applicable
Can each call have its own user attached to the session?	Yes	No (For the first call, a single "unauthenticated proxy user" is used. For the second call, a single "external proxy user" is used.)	No (a single "unauthenticated proxy user" is used for all relevant calls)
Where do authorization values come from?	not applicable (no authorization values are passed within the auth flow)	The first call includes no authorization values. For the second call, the authorization values are in the self-signed JWT provided by PolicyCenter.	not applicable (no authorization values are provided)
Does Cloud API enforce resource access?	Yes	Yes	Yes
What do the resource access IDs come from?	not applicable (no authorization values are passed within the auth flow)	PolicyCenter	not applicable (no authorization values are provided)
For more information on this auth flow	“Basic authentication” on page 27	“OAuth2 authorization code flow: Anonymous users”	“Unauthenticated callers” on page 77

part 2

Authentication flows in detail

Cloud API supports several different authentication flows. Each flow supports one of the following types of callers:

- Internal users using basic auth
- Internal users using bearer token auth
- External users
- Standalone services
- Services with user context
- Services with service account mapping
- Unauthenticated callers

This section describes each of these flows in detail.

chapter 3

Basic authentication

Within the context of system API authentication, an *internal user* is a person who is listed as a user in the ClaimCenter database. For example, Andy Applegate, a ClaimCenter adjuster, is an internal user. Internal users can use caller applications and trigger system API calls from that application. For example, suppose there is a loss documentation portal that contains pictures of a damaged auto taken by a third-party field agent. An adjuster reviews and selects pictures to be saved to ClaimCenter. This action triggers a system API call by an internal user from a caller application.

Internal users can be authenticated using either basic authentication or bearer token authentication. *Basic authentication* is an authentication method in which only the user's user name and password are provided, and they are provided in the request header.

- Internal users (and only internal users) can use basic authentication.
- With basic authentication, the authentication information is retrieved from the operational database using information in the request header

Basic authentication is useful in development when you want to test aspects of endpoint behavior that are not related to authentication. Basic authentication does not require any interaction with Guidewire Hub to generate JWTs. You can authenticate a system API call using only the caller application and ClaimCenter.

This topic describes how to implement basic authentication for internal users. (For information on how to implement bearer token authentication for internal users, see “OAuth2 authorization code flow: Internal users” on page 33.)

Overview of basic authentication

Authentication includes credentials and authorization. Authentication information for basic authentication is specified in request headers.

Credentials

With basic authentication, every internal user's credential information is stored in the ClaimCenter database.

The user name and password is provided by each caller in the request object's header.

Authorization

Endpoint access with basic authentication

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?

- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

When an internal user makes a system API call (using either basic authentication or bearer token authentication), ClaimCenter queries the operational database for this internal user's user roles. The user is given endpoint access to all API roles whose names corresponds to the names of the user's user roles.

For example, suppose that Andy Applegate is an internal user with two user roles: Adjuster and Reinsurance Manager. Andy Applegate triggers a system API call. When the API call is received, ClaimCenter queries the database for Andy's user roles. Two user roles are returned: Adjuster and Reinsurance Manager. ClaimCenter then grants Andy the endpoint access defined in the API roles named "Adjuster" and "Reinsurance Manager".

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access with basic authentication

Resource access defines, for a given type of resource, which instances of that resources the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is `claim`, but each caller may be restricted so that they can access only a subset of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies the meaning of a resource access ID. The base configuration includes the following resource access strategies for internal users:

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`cc_username`	Internal users	A ClaimCenter user name	Any information this internal user could see in ClaimCenter based on their associated Access Control Lists (ACLs).

When an internal user makes a system API call, the user name is used as the resource access ID. The `cc_username` or `pc_username` strategy is used automatically. This strategy consists of system API logic that matches, as closely as possible, the user's access as defined in the base configuration's Access Control Lists (ACLs).

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access with basic authentication

Proxy user access is not applicable to basic authentication.

Request headers

For basic authentication, authorization information is sent to ClaimCenter with the request's authorization header. The header must use this format:

```
Authorization: Basic <token>
```

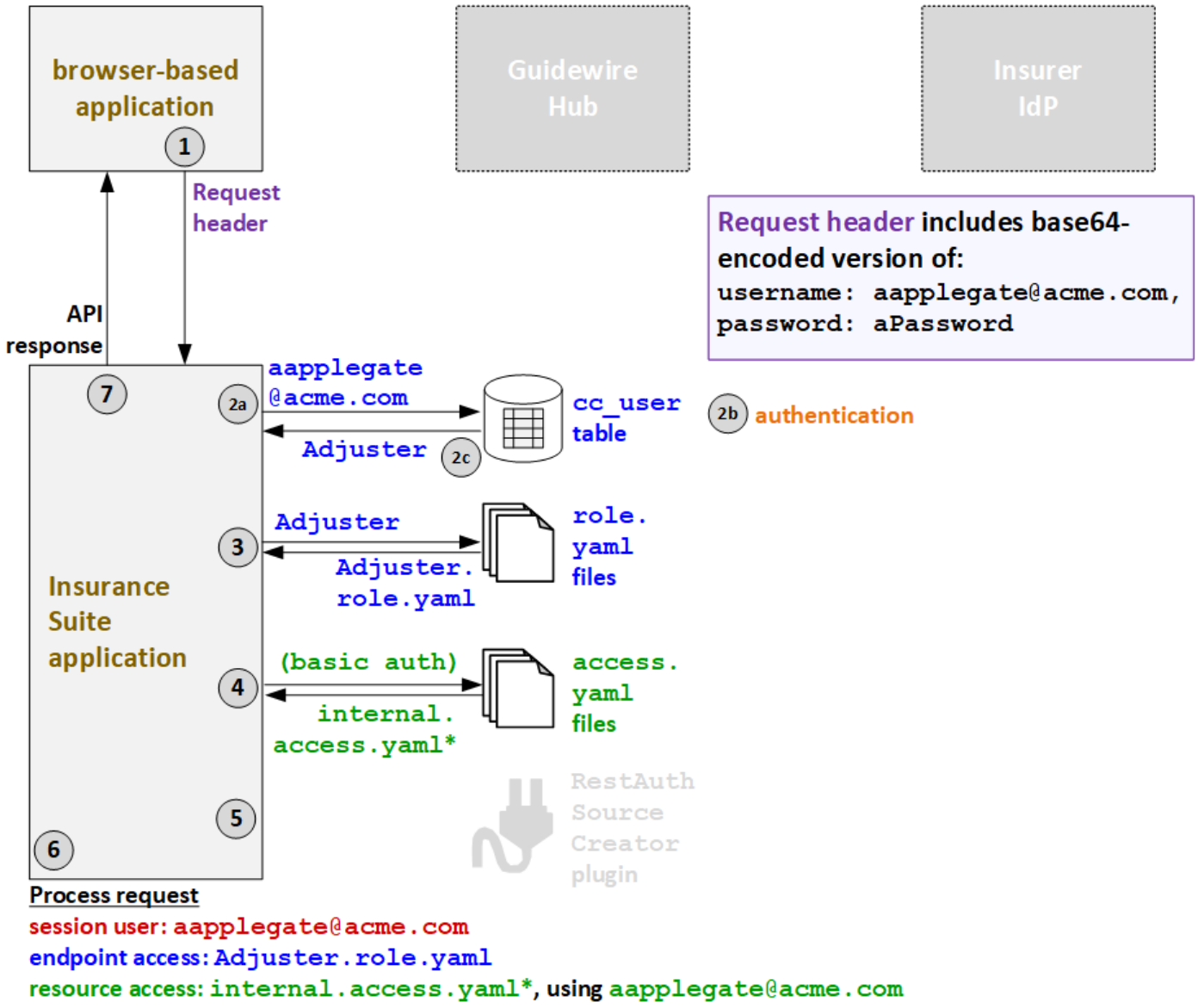
Example flow for basic authentication

The following diagram identifies the flow of authentication and authorization information for basic authentication. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by Andy Applegate, who is an internal user, using a browser-based application and basic authentication.



1. When Andy triggers an API call, the caller application sends the API request to ClaimCenter. The request header includes a base64-encoded version of the user's user name (`aapplegate@acme.com`) and password (`aPassword`).

2. ClaimCenter authenticates the user and determines the endpoint access.
   a. Using the user name in the request header (`aapplegate@acme.com`), ClaimCenter queries the user table.
   b. ClaimCenter authenticates the user by verifying that the user name and password match.
   c. ClaimCenter responds with the user roles that this user has. One role is returned: `Adjuster`.
3. Based on the returned role, the `Adjuster.role.yaml` API role file is used to define the endpoint access.
4. Next, ClaimCenter determines the resource access strategy. Because the call is using basic authentication, ClaimCenter grants resource access as defined in the `internal access.yaml` files. (* ClaimCenter starts with `internal_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with `"internal"`.)
5. Proxy user access is not relevant for basic authentication.
6. ClaimCenter processes the request.
   a. The session user is the internal user: `aapplegate@acme.com`.
   b. The endpoint access is defined by `Adjuster.role.yaml`.
   c. The resource access is defined by `internal access.yaml` using the resource access ID of `aapplegate@acme.com`.
7. ClaimCenter provides the response to the initial call.

Implementation checklist for basic authentication

To configure the system APIs for basic authentication, you may need to do the following tasks:

Task	More Information
Create or modify API roles	"Endpoint access" on page 87
Review the resource access provided to internal users	"Resource access" on page 95

To make a system API call for basic authentication, the caller application must:

1. Send the API call using basic authentication.

For more information, see "Sending authenticated calls with basic authentication" on page 30.

Sending authenticated calls with basic authentication

To request authentication for an internal user using basic authentication, the caller application must include the authentication information in the header of the request object. For example:

```
--header 'Authorization: Basic YWFwcGxlZ2F0ZTpndw=='
```

In the base configuration, when ClaimCenter receives a call with basic auth information in the header, it queries the database to verify that the user is a known internal user and that the password matches the user name. If these two things are true, the internal user is authenticated.

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

Send a Postman call with basic authentication

About this task

You can use basic authentication with API calls made from Postman.

Procedure

1. Open Postman.
2. Start a new request by clicking the + to the right of the **Launchpad** tab.
3. Specify an operation and URL as appropriate.
4. To provide authorization using basic authorization:
   a) Click the **Authorization** tab.
   b) For the **Type** drop-down list, select *Basic Auth*.
   c) In the **Username** field, enter the user name (such as `aapplegate`).
   d) In the **Password** field, enter the password (such as `gw`).
5. Click the **Send** button to the right of the request field.

Results

Every Postman tab has its own authentication information. When you modify the request on an existing tab by changing the URL or choosing a new operation, you do not need to re-enter the authentication information. But when you open a new tab, you do need to provide authentication information. If you encounter a NotFoundException such as the following example, this could be caused by not providing correct authentication information.

```
"status": 404,
"errorCode": "gw.api.rest.exceptions.NotFoundException",
"userMessage": "No resource was found at path /common/v1/activities/cc:20"
```

chapter 4

OAuth2 authorization code flow: Internal users

Within the context of system API authentication, an *internal user* is a person who is listed as a user in the ClaimCenter database. For example, Andy Applegate, a ClaimCenter adjuster, is an internal user. Internal users can use caller applications and trigger system API calls from that application. For example, suppose there is a loss documentation portal that contains pictures of a damaged auto taken by a third-party field agent. An adjuster reviews and selects pictures to be saved to ClaimCenter. This action triggers a system API call by an internal user from a caller application.

This topic describes how to implement system API authentication for internal users using bearer token authentication. (For information on how to implement authentication for internal users using basic authentication, see "Basic authentication" on page 27.)

Overview of authentication for internal users

Authentication includes credentials and authorization. Authentication information for internal users (using bearer token authentication) is specified in JWTs, and information from these JWTs is recorded in the logs.

Credentials

An internal user's credentials consist of a user name and password. This information is stored in the IdP.

When an internal user makes an API call, the caller application sends the user's credentials to Guidewire Hub. Guidewire Hub federates this information to the appropriate IdP. The IdP authenticates the user by confirming that the password is correct.

For more information on how to configure the IdP, see "Configuring the IdP" on page 84.

Authorization

Endpoint access for internal users

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

When an internal user makes a system API call (using either basic authentication or bearer token authentication), ClaimCenter queries the operational database for this internal user's user roles. The user is given endpoint access to all API roles whose names corresponds to the names of the user's user roles.

For example, suppose that Andy Applegate is an internal user with two user roles: Adjuster and Reinsurance Manager. Andy Applegate triggers a system API call. When the API call is received, ClaimCenter queries the database for Andy's user roles. Two user roles are returned: Adjuster and Reinsurance Manager. ClaimCenter then grants Andy the endpoint access defined in the API roles named "Adjuster" and "Reinsurance Manager".

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access for internal users

Resource access defines, for a given type of resource, which instances of that resources the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is `claim`, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies the meaning of a resource access ID. The base configuration includes the following resource access strategies for internal users:

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`cc_username`	Internal users	A ClaimCenter user name	Any information this internal user could see in ClaimCenter based on their associated Access Control Lists (ACLs).

When an internal user makes a system API call, the user's user name is included in the JWT. The user name is used as the resource access ID. The `cc_username` or `pc_username` strategy is used automatically. This strategy consists of system API logic that matches, as closely as possible, the user's access as defined in the base configuration's Access Control Lists (ACLs).

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access for internal users

Proxy user access is not applicable for internal users.

JWTs for internal users

JSON Web Tokens (JWTs) contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

JWTs for internal users can include the following token claims:

- `scp` - The resource access strategy to apply to the resource access ID. (For internal users, this is set to `cc_username`.)
- `cc_username` - The resource access ID. (This is the user's user name)

For example, the following JWT is for an internal user whose user name is `aapplegate`. (Information that is not relevant to system API authorization has been omitted.)

```
{
    "scp": [
        "cc_username"
    ],
    "cc_username": "aapplegate"
}
```

Note the following:

- Based on the `scp` token claim, this caller's resource access ID will be interpreted as a user name.
- Based on the `cc_username` token claim, this caller will have access to information related to what the user `aapplegate` can access.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	The user name of the internal user

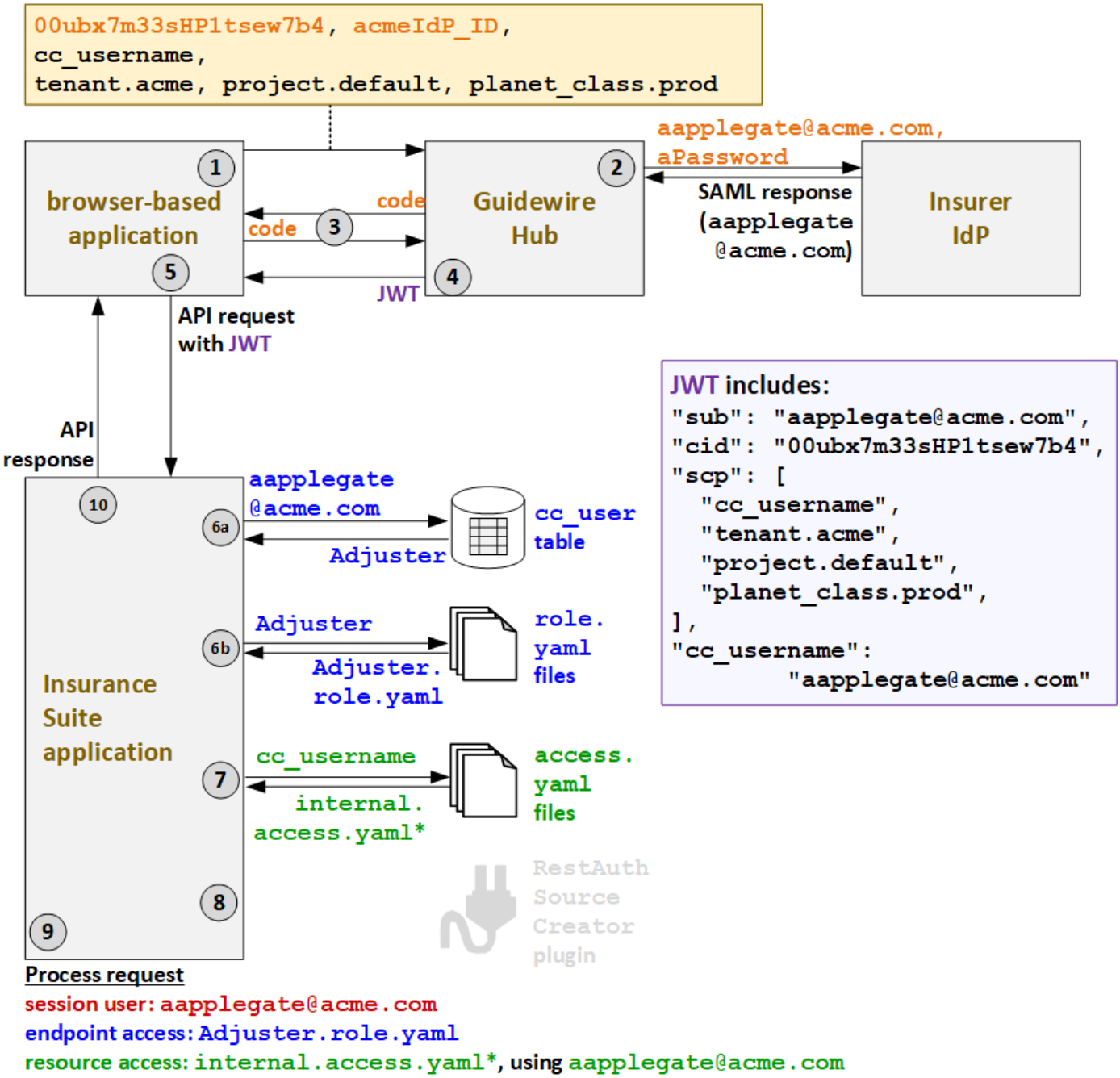
Example flow for internal users

The following diagram identifies the flow of authentication and authorization information for internal users. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by Andy Applegate, who is an internal user, using a browser-based application.



1. When Andy triggers an API call, the caller application must first request a JWT from Guidewire Hub. To initiate the process of getting the JWT, the caller application submits its client ID (`00ubx7m33sHP1tsew7b4`), the ID of the IdP (`acmeIdP_ID`), the application's resource access strategy (`cc.username`), and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. To authenticate the user, Guidewire Hub acquires the user's user name (`aapplegate@acme.com`) and password (`aPassword`) through some type of login screen. It sends this information to the appropriate IdP. The IdP authenticates the user and provides a SAML response with the user's name (`aapplegate@acme.com`).
3. Guidewire Hub sends a code to the caller application. The caller application uses this code to request a JWT.
4. Guidewire Hub generates a JWT and sends it to the caller application. This JWT includes the client ID (`cid`), a `scp` token claim which names the resource access strategy (`cc_username`) and additional deployment information, and a `cc_username` token which names the user's resource access ID (`aapplegate@acme.com`).

5. The caller application sends the API request to ClaimCenter along with the JWT.
6. ClaimCenter determines the endpoint access.
   a. Using the user name in the JWT (`aapplegate@acme.com`), ClaimCenter queries for the user roles that this user has. One role is returned: `Adjuster`.
   b. Based on the returned role, the `Adjuster.role.yaml` API role file is used to define the endpoint access.
7. Next, ClaimCenter determines the resource access strategy. Based on the resource access strategy value in the JWT (`cc_username`), it grants resource access as defined in the `internal access.yaml` files. (* ClaimCenter starts with `internal_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`internal`".)
8. Proxy user access is not relevant for internal users.
9. ClaimCenter processes the request.
   a. The session user is the internal user: `aapplegate@acme.com`.
   b. The endpoint access is defined by `Adjuster.role.yaml`.
   c. The resource access is defined by `internal access.yaml` using the resource access ID of `aapplegate@acme.com`.
10. ClaimCenter provides the response to the initial call.

Implementation checklist for internal users

To configure the system APIs for authentication for internal users (using bearer token authentication), you may need to do the following tasks:

Task	More Information
Enable asymmetric encryption	"Enabling bearer token authentication" on page 83
Provide deployment information	"Enabling bearer token authentication" on page 83
Configure the IdP to store user information	"Enabling bearer token authentication" on page 83
Register the caller application with Guidewire Hub	"Enabling bearer token authentication" on page 83
Create or modify API roles	"Endpoint access" on page 87
Review the resource access provided by the `cc_username` resource access strategy	"Resource access" on page 95

To make a system API call for internal users (using bearer token authentication), the caller application must:

1. Request a code from Guidewire Hub
2. Use the code to request a JWT from Guidewire Hub
3. Include the JWT with the system API call

For more information, see "Sending authenticated calls for internal users" on page 37.

Sending authenticated calls for internal users

When a caller application wants to make a system API call for an internal user using bearer token authentication, the caller application must:

1. Request an authorization code from Guidewire Hub
2. Use the code to request a JWT from Guidewire Hub
3. Include the JWT with the system API call

Requesting codes and JWTs from Guidewire Hub

For more information on how to request codes and JWTs from Guidewire Hub, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

Including JWTs with API calls

Once a JWT has been received from Guidewire Hub, it must be sent to ClaimCenter in the request object's Authorization header. The header must use this format:

```
Authorization: Bearer <token>
```

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

chapter 5

OAuth2 authorization code flow: External users

Within the context of system API authentication, an *external user* is a person who is known to the insurer but who is not listed as a user in the ClaimCenter database. For ClaimCenter, there are two typical types of external users:

- **Policyholders** - Users who want to interact with information about claims on their policies. For example, Ray Newton, who is a policyholder and wants to check on the status of a claim filed against his personal auto policy.
- **Service providers** - Users who want to interact with service requests. For example, Mike's Auto Detailing Shop, who wants to accept a request to repair Ray Newton's vehicle, is a service provider.

This topic describes how to implement system API authentication for external users.

> **Note:** Do not confuse service providers and services. A *service provider* is a person or business who is an external user that provides assistance to a claimant. They are often referred to as vendors. A *service* is a process or application that can execute action without direct human interaction. For more information on authentication for services, see "OAuth2 client credential flow: Standalone services" on page 47.

Overview of authentication for external users

Authentication includes credentials and authorization. Authentication information for external users is specified in JWTs, and information from these JWTs is recorded in the logs.

Credentials

An external user's credentials consist of a user name and password. This information is stored in the IdP.

When an external user makes an API call, the caller application sends the user's credentials to Guidewire Hub. Guidewire Hub federates this information to the appropriate IdP. The IdP authenticates the user by confirming that the password is correct.

For more information on how to configure the IdP, see "Configuring the IdP" on page 84.

Authorization

Endpoint access for external users

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

When an external user makes a system API call, the call includes a JWT. The JWT includes a list of one or more API roles. The user is given endpoint access to all API roles whose names correspond the roles listed in the JWT. For example, suppose that Ray Newton is a policyholder. Ray Newton triggers a system API call. The JWT identifies the Insured role. Ray Newton is given the endpoint access defined in the API role named "Insured".

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access for external users

Resource access defines, for a given type of resource, which instances of that resources the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is `claim`, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies the meaning of a resource access ID. The base configuration includes the following resource access strategies for external users:

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`cc_policyNumbers`	Account holders and policy holders	An array of policy numbers	Information associated with any claims associated with any of the policies.
`cc_gwabuid`	Claims service providers	An ABUID (Address Book Unique IDentifier) of the service provider contact	Information associated with any claims for which this user is the assigned service provider

When an external user makes a system API call, ClaimCenter checks for a resource access token claim.

- If the resource access token is `cc_policyNumbers`, the resource access IDs are treated as a list of policy numbers. The user is given access to all claims based on any of those policy numbers.
- If the resource access token is `cc_gwabuid`, the resource access ID is treated as an Address Book identifier. The user is given access to all claims where at least one of the service providers has the ID.

Cloud API requires that the JWT have no more than one resource access strategy token.

- If no resource strategy token is present, the caller is assigned the "default" resource access strategy. This resource strategy grants access to metadata endpoints only.
- If multiple resource strategy tokens are present, the call is rejected.

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access for external users

When a caller makes a system API call, the internal ClaimCenter logic may trigger checks that are unrelated to endpoint access or resource access. For example:

- A caller may attempt to assign an activity to themselves. ClaimCenter must check to see if the caller has sufficient permission to own an activity.

- A caller may attempt to create a payment for $2000. ClaimCenter must check to see if the amount of the payment exceeds the caller's authority limit.

External users are not listed in the ClaimCenter operational database, and therefore do not have any system permissions or authority limits tied to them. To execute these checks, the system APIs make use of proxy users. A *proxy user* is an internal user that is assigned to an external user or service when the API call is made. Whenever internal ClaimCenter logic must check to see if the caller has sufficient access, the proxy user is checked.

For more information on how proxy user access behaves, see “Proxy user access” on page 101.

JWTs for external users

JSON Web Tokens (JWTs) contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

JWTs for external users can include the following token claims:

- `groups` - The API roles to assign to the external user.
  - In this token claim, the group name is prefixed by "`gwa.<planetclass>.<xc>.`", where `<planetclass>` is set to either "`prod`", "`preprod`", or "`lower`", and where <*xc*> is the application code (such as "cc" or "pc").
- `scp` - The resource access strategy to apply to the resource access IDs
- `cc_policyNumbers` - The resource access IDs (when `scp` includes `cc_policyNumbers`)
- `cc_gwabuid` - The resource access IDs (when `scp` includes `cc_gwabuid`)

For example, the following JWT is for a external user who is a policyholder with two policies: 54-123456 and 54-273411. (Information that is not relevant to system API authorization has been omitted.)

```
{
    "groups" : [
        "gwa.prod.cc.Insured"
    ],
    "scp": [
        "cc_policyNumbers"
    ],
    "cc_policyNumbers": [
        "54-123456",
        "54-273411"
    ]
}
```

Note the following:

- Based on the `groups` token claim, this caller will be given endpoint access as defined in the role named "Insured".
- Based on the `scp` token claim, this caller's resource access IDs will be interpreted as policy numbers.
- Based on the `cc_policyNumbers` token claim, this caller will have access to information related to claims on either policy 54-123456 or policy 54-273411.

As a second example, the following JWT is for a external user who is a vendor. The vendor's Guidewire Address Book unique identifier is cc:demo_4532. (Information that is not relevant to system API authorization has been omitted.)

```
{
    "groups" : [
        "gwa.prod.cc.ServiceRequestSpecialist"
    ],
    "scp": [
        "cc_gwabuid"
    ],
    "cc_gwabuid": [
        "cc:demo_4532"
    ]
}
```

Note the following:

- Based on the `groups` token claim, this caller will be given endpoint access as defined in the role named "ServiceRequestSpecialist".

- Based on the `scp` token claim, this caller's resource access IDs will be interpreted as a Guidewire Address Book unique identifier.
- Based on the `cc_gwabuid` token claim, this caller will have access to information related to claims where one of the service providers is a contact whose ID is cc:demo_4532.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	The user name of the external user

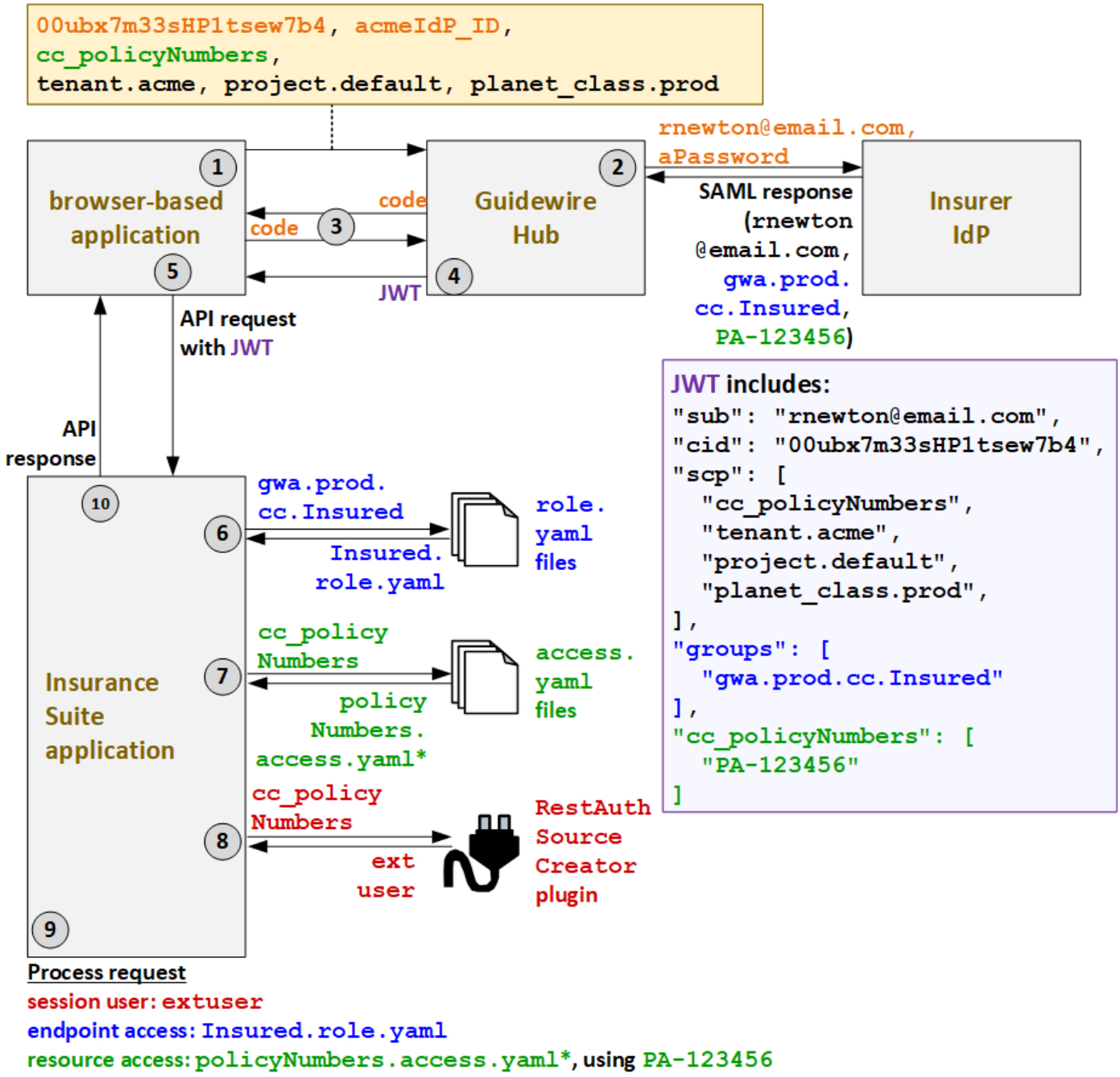
Example flow for external users

The following diagram identifies the flow of authentication and authorization information for external users. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by Ray Newton, who is an external user, using a browser-based application.



1. When Ray triggers an API call, the caller application must first request a JWT from Guidewire Hub. To initiate the process of getting the JWT, the caller application submits its client ID (`00ubx7m33sHP1tsew7b4`), the ID of the IdP (`acmeIdP_ID`), the application's resource access strategy (`cc.username`), and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. To authenticate the user, Guidewire Hub acquires the user's user name (`rnewton@email.com`) and password (`aPassword`) through some type of login screen. It sends this information to the appropriate IdP. The IdP authenticates the user and provides a SAML response with the user's name (`rnewton@email.com`), the user's groups (`gwa.prod.cc.Insured`), and the user's resource access IDs. For Ray, this is a list of his policy numbers (`PA-123456`).
3. Guidewire Hub sends a code to the caller application. The caller application uses this code to request a JWT.

4. Guidewire Hub generates a JWT and sends it to the caller application. This JWT includes the client ID (`cid`), a `scp` token claim which names the resource access strategy (`cc_policyNumbers`) and additional deployment information, a `groups` token which names the user's groups (`gwa.prod.cc.Insured`), and a `cc_policyNumbers` token which names the user's resource access IDs (`PA-123456`).
5. The caller application sends the API request to ClaimCenter along with the JWT.
6. ClaimCenter determines the endpoint access. Based on the groups listed in the JWT (`gwa.prod.cc.Insured`), the `Insured.role.yaml` API role file is used to define the endpoint access.
7. Next, ClaimCenter determines the resource access strategy. Based on the resource access strategy value in the JWT (`cc_policyNumbers`), it grants resource access as defined in the `policyNumbers access.yaml` files. (* ClaimCenter starts with `policyNumbers_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`policyNumbers`".)
8. To determine which proxy user to assign to the session, ClaimCenter calls the `RestAuthenticationSourceCreator` plugin. The JWT specified a resource access strategy of `cc_policyNumbers`. So, the plugin returns the proxy user for external users: `extuser`.
9. ClaimCenter processes the request.
   a. The session user is the proxy external user: `extuser`.
   b. The endpoint access is defined by `Insured.role.yaml`.
   c. The resource access is defined by `policyNumbers access.yaml` using the resource access ID of `PA-123456`.
10. ClaimCenter provides the response to the initial call.

Implementation checklist for external users

To configure the system APIs for authentication for external users, you may need to do the following tasks:

Task	More Information
Enable asymmetric encryption	"Enabling bearer token authentication" on page 83
Provide deployment information	"Enabling bearer token authentication" on page 83
Configure the IdP to store user information	"Enabling bearer token authentication" on page 83
Register the caller application with Guidewire Hub	"Enabling bearer token authentication" on page 83
Create or modify API roles	"Endpoint access" on page 87
Review the resource access provided by the `cc_policyNumbers` and `cc_gwabuid` resource access strategies	"Resource access" on page 95
Configure the proxy user	"Proxy user access" on page 101

To make a system API call for external users, the caller application must:

1. Request a code from Guidewire Hub
2. Use the code to request a JWT from Guidewire Hub
3. Include the JWT with the system API call

For more information, see "Sending authenticated calls for external users" on page 44.

Sending authenticated calls for external users

When a caller application wants to make a system API call for an external user, the caller application must:

1. Request an authorization code from Guidewire Hub
2. Use the code to request a JWT from Guidewire Hub
3. Include the JWT with the system API call

Requesting codes and JWTs from Guidewire Hub

For more information on how to request codes and JWTs from Guidewire Hub, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

Including JWTs with API calls

Once a JWT has been received from Guidewire Hub, it must be sent to ClaimCenter in the request object's Authorization header. The header must use this format:

```
Authorization: Bearer <token>
```

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

chapter 6

OAuth2 client credential flow: Standalone services

A *service* is an application that typically executes action without human intervention. Services typically have no user interface. Examples of services include:

- An FNOL application that takes a collection of First Notice of Loss information stored in a database or file and sends it to ClaimCenter so that claims can be created.
- A Metro Report application that provides ClaimCenter with information about police reports related to auto accidents.
- An application that uploads pictures of a covered location or vehicle, either when a policy is bound or after a loss has occurred.

This topic discusses how to execute authentication for standalone services.

Authentication options for services

There are several ways a service can execute authentication with Cloud API.

Standalone service

A service can authenticate **as a standalone service**. In this case, the service executes the call as itself. It does not execute the call as a specific person or on behalf of a specific person. The service does not execute the call using a service account stored in ClaimCenter.

ClaimCenter designates a single internal user as the "proxy service user" for all standalone service calls. This proxy service user is attached to the standalone service session. If the call creates or modifies an object, the proxy service user is recorded as the user of record.

The primary advantage to this approach is that you need to manage authentication and authorization information at the service level only. There is no need to create and manage user accounts, user permissions, or additional mappings.

The primary disadvantage is that all standalone service calls share a single proxy service user. When a standalone service call creates or modifies an object, it may not be possible to identify which service made the call.

Service with user context

A service can authenticate **with user context**. In this case, the service presents information about itself. The call also includes a `GW-User-Context` header that provides information about a specific user. The user does not necessarily exist

in the ClaimCenter database. The call is able to do only the things that both the service by itself could do and the user by itself could do.

The specified user can be an internal user (a user who is listed in the ClaimCenter database). When this is the case, this internal user is attached to the session. If the call creates or modifies an object, this internal user is recorded as the user of record.

The specified user can be an external user (a user who is not listed in the ClaimCenter database). ClaimCenter designates a single internal user as the "proxy external user" for all service with user context calls that reference external users. When the specified user is an external user, the external proxy user is attached to the session. If the call creates or modifies an object, the external proxy user is recorded as the user of record.

The primary advantage to this approach is that a single service can send calls on behalf of different users. At the service level, you can specify service-level access. But, you can also further control access for each associated user.

There are two primary disadvantages. First, you must maintain access information at two levels: the service level and at the user level. Second, a service can specify any user in its header. There is no way to restrict a given set of users for use by a given service.

Service with service account mapping

A service can authenticate **with service account mapping**. In this case, the service is automatically mapped to a "service account". The mapping information is specified elsewhere in the environment. The service account is a user account in the ClaimCenter database that is intended to be used only by the service and not by any person.

The primary advantage to this approach is that permissions and auditing for the call is tied to a service account that is listed in the ClaimCenter database. You can create a different service account for each service and have fine control over the permissions available to each service.

The primary disadvantage is that you must create and maintain service accounts in ClaimCenter for calls made by external applications. Also, you must maintain the mapping information that maps each service to its service account.

Comparing the different approaches

The following table compares each approach.

	Standalone service	Service with user context	Service with service account mapping
Does the call provide information about the service?	Yes, in the JWT.	Yes, in the JWT.	Yes, in the JWT.
Does there need to be a user account in the ClaimCenter database for the call?	No	If the associated user is an internal user, yes. If the associated user is an external user, no.	Yes. (This user account is the "service account".)
Does the call include information about a user or user account?	No	Yes, in the `GW-User-Context` header.	No. The call provides a client ID for the service, but the mapping of client ID to service account is stored elsewhere.
Which endpoints can the call access?	The endpoints available to the service's API roles	The endpoints available to both the service's API roles and the user's API roles.	The endpoints available to the service account.
Which resources can the call access?	All resources (in the base configuration).	The resources available to both the service and the user.	The resources available to the service account.
What is the session user set to?	The proxy service user.	If the associated user is an internal user, the internal user. If the associated user is an external user, the proxy external user.	The service account.

This topic focuses on authentication for standalone services.

- For more information on authentication for services with user context, see "OAuth2 client credential flow: Services with user context" on page 55.
- For more information on authentication for services with service account mapping, see "OAuth2 client credential flow: Services with service account mapping" on page 67.

Overview of authentication for standalone services

Authentication includes credentials and authorization. Authentication information for services is specified in JWTs, and information from these JWTs is recorded in the logs.

Credentials

When a service makes an API call, the service sends a client ID and secret to Guidewire Hub. Guidewire Hub authenticates the service by confirming that the client secret is correct. This is true for standalone services, services with user context, and services with service account mapping.

For more information on how client IDs and secrets are registered with Guidewire Hub, see "Registering the caller application with Guidewire Hub" on page 85.

Authorization

Endpoint access for standalone services

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

Theoretically, a standalone service can be associated with multiple API roles. Typically, insurers create one API role for each service and this role is used only by this service.

For a standalone service call, the system APIs check the API role or roles assigned to the service. The call has access to the endpoints, operations, and fields specified in those roles. For example, suppose that the ACME External Document Manager service has the following API role with the following endpoint access:

- `acme_externaldocumentmanager`
  - GET `/documents`
  - POST `/documents`

Then, suppose ACME External Document Manager service makes a standalone service call. The call would have access to GET `/documents` and POST `/documents`. But if there was a DELETE `/documents` endpoint, the call would not have access to it because it has not been specified in the `acme_externaldocumentmanager` role.

For more information on how API roles are configured, see "Endpoint access" on page 87.

Resource access for standalone services

Resource access defines, for a given type of resource, which instances of that resource the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is `claim`, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.

- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies which resources a caller can access. The base configuration includes resource access strategies for standalone services. But, these strategies do not involve resource access IDs or restrict the service in any way. Once a service is given access to a set of endpoints, the service can access any resource available to those endpoints. These applications are expected to be configured such that they access only the resources appropriate for the circumstance.

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`cc.service`	Services	Not applicable	All resources

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access for standalone services

Every Cloud API call occurs within the context of a session. A user listed in the ClaimCenter database must be attached to this session. ClaimCenter can use this "session user" in different ways.

- If the call creates or modifies an object, the session user is recorded as the `CreateUser` or `UpdateUser` of that object.
- If the call triggers an authority limit check, the session user's authority limits are checked.
- There is a small chance the call could trigger logic that must check to see if the caller has a specific domain-level system permission, such as the permission to own an activity. When this occurs, the session user's system permissions are checked.

A *proxy user* is an internal user that is assigned to a session for an API call made by an external user or service. Proxy users are assigned by the `RestAuthenticationSourceCreatorPlugin` plugin. This plugin specifies four proxy users. One of them, the proxy service user, is used for calls made by standalone services.

- If the call creates or modifies an object, the proxy service user is recorded as the `CreateUser` or `UpdateUser` of that object.
- If the call triggers an authority limit or domain-level system permission, the limits and permissions of the proxy service user are checked.

For more information on proxy users, see “Proxy user access” on page 101.

JWTs for standalone services

JSON Web Tokens (JWTs) contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

When a service makes a standalone service call, only some of the information in the JWT is specific to Cloud API authentication. The following are the token claims in a JWT for a standalone service call that pertain to Cloud API authentication.

```
"sub": "<clientId>",
"cid": "<clientId>",
"scp": [
  "cc.service",
  "scp.cc.<serviceAPIRole>"
]
```

- `sub` is the subject of the token. This is set to the service's client ID.
- `cid` is the client ID of the service. This is also set to the service's client ID.
- The `scp` token claim has at least the following entries:
  - The `cc.service` value, which specifies the caller is a service.

- A list of one or more API roles associated with the service. These roles are prefixed with "`scp.cc.`".

For example, the following JWT is for ACME's external FNOL reporter service. (Information that is not relevant to Cloud API authorization has been omitted.)

```
{
    "sub": "acme_externalfnolreporter",
    "cid": "acme_externalfnolreporter",
    "scp": [
        "cc.service",
        "scp.cc.acme_externalfnolreporter"
    ]
}
```

Note the following:

- Based on the `scp` token claim, this caller will be given endpoint access as defined in the role named "acme_externalfnolreporter". Because of the `cc.service` entry, this caller uses a resource access strategy that provides access to all resources.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	An empty string

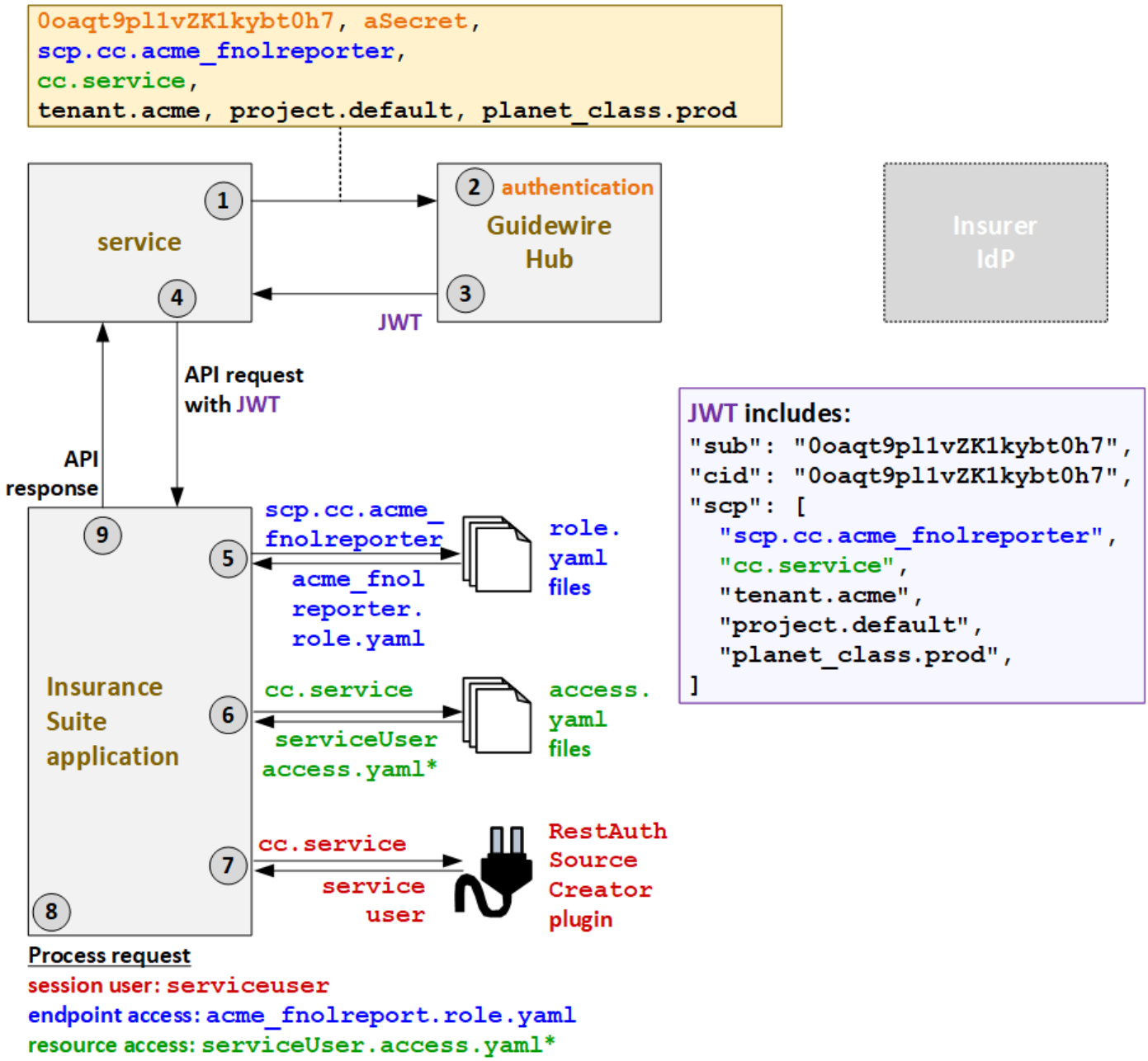
Example flow for standalone services

The following diagram identifies the flow of authentication and authorization information for standalone services. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by the Acme FNOLReporter.



1. When FNOLReporter triggers an API call, it must first request a JWT from Guidewire Hub. The request for the JWT includes the client ID (`0oaqt9pl1vZK1kybt0h7`), the secret (`aSecret`), the application's API role (`scp.cc.acme_fnolreporter`), the application's resource access strategy (`cc.service`), and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. Guidewire Hub authenticates the services based on the client ID and secret. It also verifies that the API role and resource access strategy provided in the request match what was specified when the service was registered with Guidewire Hub.
3. Guidewire Hub generates a JWT and sends it to the service. This JWT includes the client ID (`cid`) and a `scp` token claim which names the API role (`scp.cc.acme_fnolreporter`), the resource access strategy (`cc.service`), and additional deployment information.
4. The service sends the API request to ClaimCenter along with the JWT.

5. ClaimCenter determines the endpoint access. Based on the "`scp.cc.`" value listed in the JWT (`scp.cc.acme_fnolreporter`), the `acme_fnolreporter.role.yaml` API role file is used to define the endpoint access.
6. Next, ClaimCenter determines the resource access strategy. Based on the resource access strategy value in the JWT (`cc.service`), it grants resource access as defined in the `serviceUser access.yaml` files. (* ClaimCenter starts with `serviceUser_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`serviceUser`".)
7. To determine which proxy user to assign to the session, ClaimCenter calls the `RestAuthenticationSourceCreator` plugin. The JWT specified a resource access strategy of `cc.service`. So, the plugin returns the proxy user for services: `serviceuser`.
8. ClaimCenter processes the request.
   a. The session user is the proxy service user: `serviceuser`.
   b. The endpoint access is defined by `acme_fnolreporter.role.yaml`.
   c. The resource access is defined by `serviceUser access.yaml`. In the base configuration, the `serviceuser access.yaml` files make all resources available. Therefore, logically speaking, there are no resource access restrictions.
9. ClaimCenter provides the response to the initial call.

Implementation checklist for standalone services

To configure the system APIs for authentication for standalone services, you may need to do the following tasks:

Task	More Information
Enable asymmetric encryption	"Enabling bearer token authentication" on page 83
Provide deployment information	"Enabling bearer token authentication" on page 83
Register the caller application with Guidewire Hub	"Enabling bearer token authentication" on page 83
Create or modify API roles	"Endpoint access" on page 87
Configure the proxy user	"Proxy user access" on page 101

Sending authenticated calls for standalone services

When a caller application wants to make a system API call for a standalone service, the caller application must:

1. Request a JWT from Guidewire Hub
2. Include the JWT with the system API call

Requesting codes and JWTs from Guidewire Hub

For more information on how to request JWTs from Guidewire Hub, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

Including JWTs with API calls

Once a JWT has been received from Guidewire Hub, it must be sent to ClaimCenter in the request object's Authorization header. The header must use this format:

```
Authorization: Bearer <token>
```

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

chapter 7

OAuth2 client credential flow: Services with user context

A *service* is an application that typically executes action without human intervention. Services typically have no user interface. Examples of services include:

- An FNOL application that takes a collection of First Notice of Loss information stored in a database or file and sends it to ClaimCenter so that claims can be created.
- A Metro Report application that provides ClaimCenter with information about police reports related to auto accidents.
- An application that uploads pictures of a covered location or vehicle, either when a policy is bound or after a loss has occurred.

This topic discusses how to execute authentication for services with user context.

Authentication options for services

There are several ways a service can execute authentication with Cloud API.

Standalone service

A service can authenticate **as a standalone service**. In this case, the service executes the call as itself. It does not execute the call as a specific person or on behalf of a specific person. The service does not execute the call using a service account stored in ClaimCenter.

ClaimCenter designates a single internal user as the "proxy service user" for all standalone service calls. This proxy service user is attached to the standalone service session. If the call creates or modifies an object, the proxy service user is recorded as the user of record.

The primary advantage to this approach is that you need to manage authentication and authorization information at the service level only. There is no need to create and manage user accounts, user permissions, or additional mappings.

The primary disadvantage is that all standalone service calls share a single proxy service user. When a standalone service call creates or modifies an object, it may not be possible to identify which service made the call.

Service with user context

A service can authenticate **with user context**. In this case, the service presents information about itself. The call also includes a `GW-User-Context` header that provides information about a specific user. The user does not necessarily exist

in the ClaimCenter database. The call is able to do only the things that both the service by itself could do and the user by itself could do.

The specified user can be an internal user (a user who is listed in the ClaimCenter database). When this is the case, this internal user is attached to the session. If the call creates or modifies an object, this internal user is recorded as the user of record.

The specified user can be an external user (a user who is not listed in the ClaimCenter database). ClaimCenter designates a single internal user as the "proxy external user" for all service with user context calls that reference external users. When the specified user is an external user, the external proxy user is attached to the session. If the call creates or modifies an object, the external proxy user is recorded as the user of record.

The primary advantage to this approach is that a single service can send calls on behalf of different users. At the service level, you can specify service-level access. But, you can also further control access for each associated user.

There are two primary disadvantages. First, you must maintain access information at two levels: the service level and at the user level. Second, a service can specify any user in its header. There is no way to restrict a given set of users for use by a given service.

Service with service account mapping

A service can authenticate **with service account mapping**. In this case, the service is automatically mapped to a "service account". The mapping information is specified elsewhere in the environment. The service account is a user account in the ClaimCenter database that is intended to be used only by the service and not by any person.

The primary advantage to this approach is that permissions and auditing for the call is tied to a service account that is listed in the ClaimCenter database. You can create a different service account for each service and have fine control over the permissions available to each service.

The primary disadvantage is that you must create and maintain service accounts in ClaimCenter for calls made by external applications. Also, you must maintain the mapping information that maps each service to its service account.

Comparing the different approaches

The following table compares each approach.

	Standalone service	Service with user context	Service with service account mapping
Does the call provide information about the service?	Yes, in the JWT.	Yes, in the JWT.	Yes, in the JWT.
Does there need to be a user account in the ClaimCenter database for the call?	No	If the associated user is an internal user, yes. If the associated user is an external user, no.	Yes. (This user account is the "service account".)
Does the call include information about a user or user account?	No	Yes, in the `GW-User-Context` header.	No. The call provides a client ID for the service, but the mapping of client ID to service account is stored elsewhere.
Which endpoints can the call access?	The endpoints available to the service's API roles	The endpoints available to both the service's API roles and the user's API roles.	The endpoints available to the service account.
Which resources can the call access?	All resources (in the base configuration).	The resources available to both the service and the user.	The resources available to the service account.
What is the session user set to?	The proxy service user.	If the associated user is an internal user, the internal user. If the associated user is an external user, the proxy external user.	The service account.

This topic focuses on authentication for services with user context.

- For more information on authentication for standalone services, see "OAuth2 client credential flow: Standalone services" on page 47.
- For more information on authentication for services with service account mapping, see "OAuth2 client credential flow: Services with service account mapping" on page 67.

Overview of authentication for services with user context

Authentication includes credentials and authorization. Authentication information for services is specified in JWTs, and information from these JWTs is recorded in the logs.

Credentials

When a service makes an API call, the service sends a client ID and secret to Guidewire Hub. Guidewire Hub authenticates the service by confirming that the client secret is correct. This is true for standalone services, services with user context, and services with service account mapping.

When a service authenticates with user context, it provides information about a user. However, there is no authentication at the user level. Authentication occurs only at the service level.

For more information on how client IDs and secrets are registered with Guidewire Hub, see "Registering the caller application with Guidewire Hub" on page 85.

Authorization

Endpoint access for services with user context

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

For a service-with-user-context call, the system APIs check two sets of API roles:

- The API roles assigned to the service
- The API roles assigned to the user

The endpoint access granted to the call is the intersection of the endpoint access granted by these two sets. In other words, in order to access an endpoint, operation, or field, the access must be granted to at least one API role assigned to the service and at least one API role assigned to the user account.

For example, suppose that the ACME External Document Manager service has the following API role with the following endpoint access:

- `acme_externaldocumentmanager`
  - GET `/documents`
  - POST `/documents`

And, suppose that Ray Newton has the following API role with the following endpoint access:

- `Insured`
  - GET `/documents`
  - GET `/coverages`

Suppose ACME External Document Manager service makes a call as a service with user account using the Ray Newton user account. The call would have access to GET `/documents`, as this endpoint has been granted to both the

service and the user account. The call would not have access to either POST `/documents` or GET `/coverages`, as neither of these endpoints have been granted to both the service and the user account.

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access for services with user context

Resource access defines, for a given type of resource, which instances of that resource the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is claim, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies which resources a caller can access. The base configuration includes resource access strategies for users and services.

- Resource access strategies for users typically restrict the caller to objects owned by the user. For example, an account holder can see objects owned (directly or indirectly) by the account.
- Resource access strategies for services typically do not restrict the caller in any way.

Similar the endpoint access, the resource access for a service-with-user-context call is the intersection of the resource access for the service and the resource access for the user. In the base configuration, services have access to all resources. Therefore, the resource access for a service-with-user-context call is logically equivalent to the resource access for the user.

For example, suppose that the ACME External Document Manager service has access to the following documents:

- (all documents)

And, suppose that Ray Newton has access to the following documents:

- Documents associated with policy 55-123456
  - Document xc:127
  - Document xc:356
- Documents associated directly with account C000324667
  - Document xc:888

If ACME External Document Manager service makes a service-with-user-account call for Ray Newton, the call would have access to document xc:127, xc:356, and xc:888, as these resources can be accessed by both the service and the user. The call would not have access to any other documents, as there are no other documents that both the service and the user have access to.

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access for services with user context

Every Cloud API call occurs within the context of a session. A user listed in the ClaimCenter database must be attached to this session. ClaimCenter can use this "session user" in different ways.

- If the call creates or modifies an object, the session user is recorded as the `CreateUser` or `UpdateUser` of that object.
- If the call triggers an authority limit check, the session user's authority limits are checked.
- There is a small chance the call could trigger logic that must check to see if the caller has a specific domain-level system permission, such as the permission to own an activity. When this occurs, the session user's system permissions are checked.

When a service makes a call with a user context, the associated user could be an internal user. An *internal user* is a person who is listed as a user in the ClaimCenter operational database. In this case, that internal user is used as the session user.

When a service makes a call with a user context, the associated user could be an external user. An *external user* is a person who is known to the insurer but who is not listed as a user in the ClaimCenter operational database. For example, this could be a policyholder, vendor, or account holder. When the user context specifies an external user, a proxy user must be assigned to the session.

A *proxy user* is an internal user that is assigned to a session for an API call made by an external user or service. Proxy users are assigned by the `RestAuthenticationSourceCreatorPlugin` plugin. This plugin specifies four proxy users. One of them, the proxy external user, is used for calls made either by external users or by services with user context where the user context specifies an external user.

For more information on proxy users, see “Proxy user access” on page 101.

JWTs for services with user context

JSON Web Tokens (JWTs) contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

When a service makes a call with user context, only some of the information in the JWT is specific to Cloud API authentication. The following are the token claims in a JWT for a service-for-user-context call that pertain to Cloud API authentication.

```
"sub": "<clientId>",
"cid": "<clientId>",
"scp": [
  "cc.service",
  "scp.cc.<serviceAPIRole>",
  "cc.allowusercontext"
]
```

- `sub` is the subject of the token. This is set to the service's client ID.
- `cid` is the client ID of the service. This is also set to the service's client ID.
- The `scp` token claim has at least the following entries:
  - The `cc.service` value, which specifies the caller is a service.
  - A list of one or more API roles associated with the service. These roles are prefixed with "`scp.cc.`".
  - The `cc.allowusercontext` value, which allows the call to specify additional user context in the header. (If the header contains a `GW-User-Context` header, then this is treated as a service-with-user-context call and not a standalone service call).

The `cc.allowusercontext` value indicates that the call is also presenting a user context. Information about the user is specified in a `GW-User-Context` header. For internal users, the header must specify the user name and the resource access strategy and resource access IDs. For external users, the header must specify the user name, the user roles, and the resource access strategy and resource access IDs. The header must be base64-encoded.

The header must be a JSON payload that is formatted as described in the following paragraphs.

For an internal user, the syntax of the `GW-User-Context` header is:

```
{
  "sub": "<userName>",
  "cc_username" : "<userName>"
}
```

Note the following:

- The user name is specified in the `sub` token claim.
- The resource access strategy is specified by the presence of the `cc_username` token claim.
- The resource access ID is the user's name, which is specified within the `cc_username` token claim.

For an external user who is a policyholder, the syntax of the `GW-User-Context` header is:

```
{
  "sub": "<userName>",
```

```
  "groups": [
    "<userAPIroleList>"
  ],
  "cc_policyNumbers" : [
    "<policyNumbers>"
  ]
}
```

Note the following:

- The user name is specified in the `sub` token claim. (This is used for logging, but otherwise it has no functional impact.)
- The user's API roles are listed in the `groups` token claim. Every role name must be prefixed with "`gwa.<planet_class>.cc.`".
- The resource access strategy is specified by the presence of the `cc_policyNumbers` token claim.
- The resource access IDs are a list of policy numbers, which are specified within the `cc_policyNumbers` token claim.

For an external user who is a service provider (also referred to as a vendor), the syntax of the `GW-User-Context` header is:

```
{
  "sub": "<userName>",
  "groups": [
    "<userAPIroleList>"
  ],
  "cc_gwabuid" : "<addressBookUniqueIdentifier>"
}
```

Note the following:

- The user name is specified in the `sub` token claim. (This is used for logging, but otherwise it has no functional impact.)
- The user's API roles are listed in the `groups` token claim. Every role name must be prefixed with "`gwa.<planet_class>.cc.`".
- The resource access strategy is specified by the presence of the `cc_gwabuid` token claim.
- The resource access ID is a GuideWire Address Book Unique IDentified, which is specified within the `cc_gwabuid` token claim.

  Note: If a call includes a JWT with the `cc.allowusercontext` token claim, but the request object's header does not contain a user context header, Cloud API treats the call as if it were coming from a standalone service. In other words, the call will be restricted to the access provided to the service. No user-based restrictions are applied because there was no user context header specifying a user.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	If the user in the user context is an internal user, the user name of the internal user. If the user in the user context is an external user, the `sub` value from the user context.

Example flow for services with user context

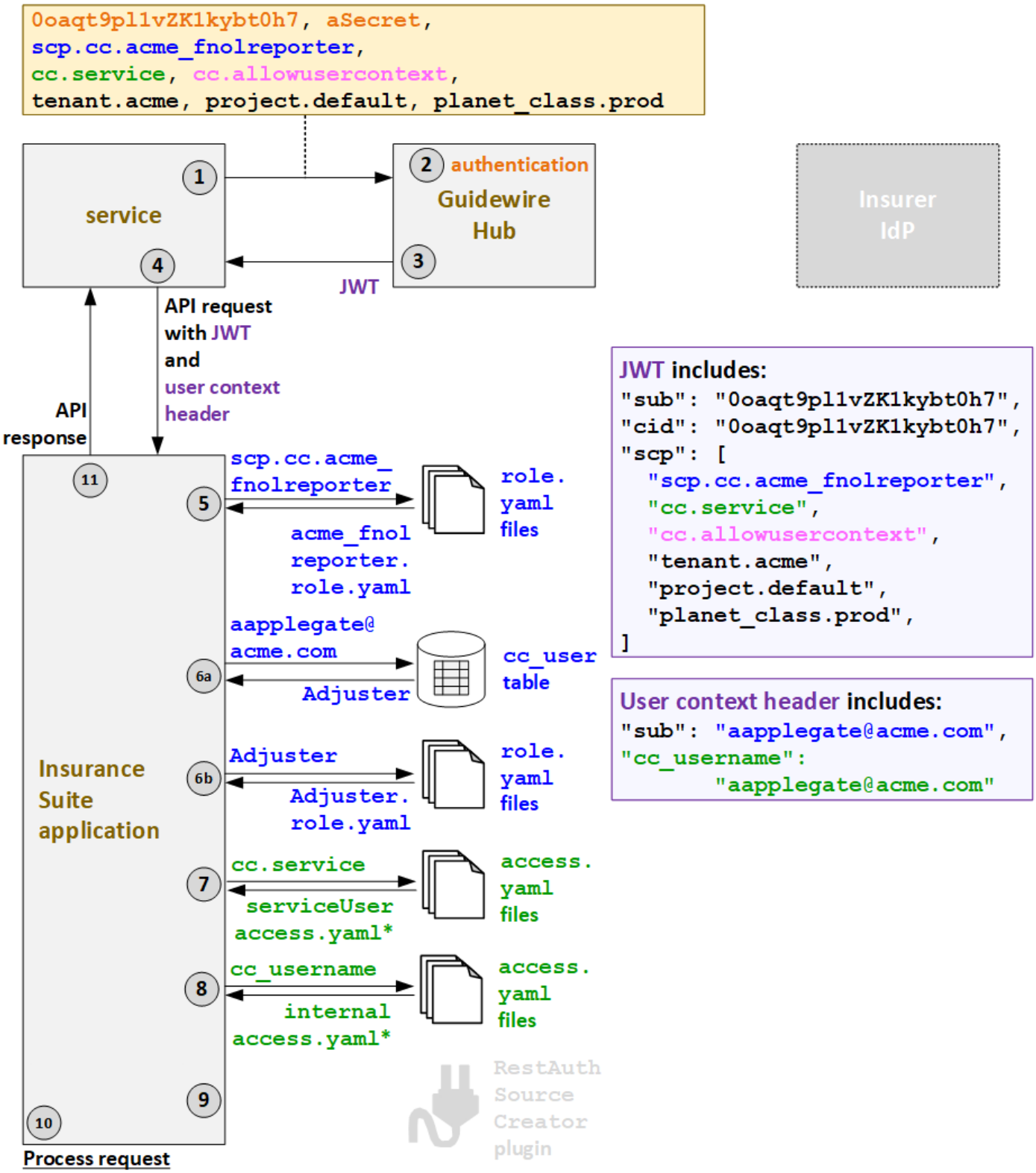
The following diagram identifies the flow of authentication and authorization information for services with user context. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

Services with user context: internal users

In the following example, an API call is triggered by the Acme FNOLReporter service on behalf of Andy Applegate, who is an internal user.



Process request

session user: aapplegate@acme.com

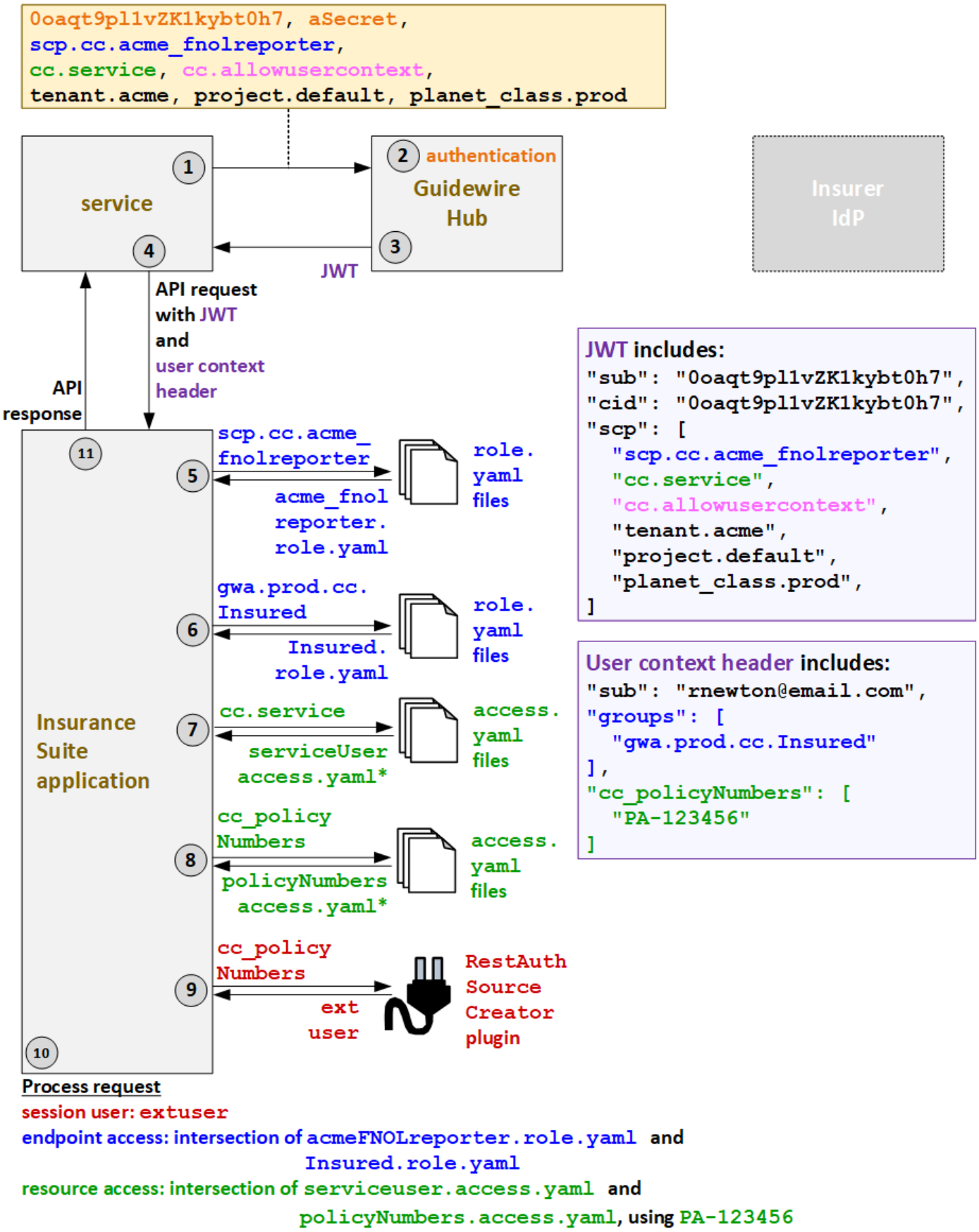
endpoint access: intersection of acmeFNOLreporter.role.yaml and Adjuster.role.yaml

resource access: intersection of serviceuser.access.yaml and internal.access.yaml, using aapplegate@acme.com

1. When FNOLReporter triggers an API call, it must first request a JWT from Guidewire Hub. The request for the JWT includes the client ID (`0oaqt9pl1vZK1kybt0h7`), the secret (`aSecret`), the application's API role (`scp.cc.acme_fnolreporter`), the application's resource access strategy (`cc.service`), the fact that the call will be made for a user (`cc.allowusercontext`) and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. Guidewire Hub authenticates the services based on the client ID and secret. It also verifies that the API role and resource access strategy provided in the request match what was specified when the service was registered with Guidewire Hub.
3. Guidewire Hub generates a JWT and sends it to the service. This JWT includes the client ID (`cid`) and a `scp` token claim which names the API role (`scp.cc.acme_fnolreporter`), the resource access strategy (`cc.service`), the `cc.allowusercontext` value (which indicates that user information is specified in an additional user context header), and additional deployment information.
4. The service sends the API request to ClaimCenter along with the JWT and a user context header that identifies the user (`aapplegate@acme.com`), the user's resource access strategy (`cc_username`), and resource access ID (`aapplegate@acme.com`).
5. ClaimCenter must determine the endpoint access at both the service level and the user level. It starts at the service level. Based on the API role value in the JWT (`scp.cc.acme_fnolreporter`), the `acme_fnolreporter.role.yaml` API role file is used to define the service-level access.
6. Next, ClaimCenter determines the user-level endpoint access.
   a. Using the user name in the user context header (`aapplegate@acme.com`), ClaimCenter queries for the user roles that this user has. One role is returned: `Adjuster`.
   b. Based on the returned role, the `Adjuster.role.yaml` API role file is used to define the user-level access.
7. ClaimCenter must also determine the resource access at the service level and the user level. It starts with the service-level resource access strategy. Based on the resource access strategy value in the JWT (`cc.service`), it grants service-level resource access as defined in the `serviceUser access.yaml` files. (* ClaimCenter starts with `serviceUser_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`serviceUser`".)
8. ClaimCenter determines the user-level resource access strategy. Based on the resource access strategy value in the user context header (`cc_username`), it grants user-level resource access as defined in the `internal access.yaml` files. (* ClaimCenter starts with `internal_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`internal`".)
9. Proxy user access is not relevant for services with user context when the user is an internal user.
10. ClaimCenter processes the request.
    a. The session user is the internal user: `aapplegate@acme.com`.
    b. The endpoint access is the intersection of the endpoints and operations defined granted at the service level (`acme_fnolreporter.role.yaml`) and at the user level (`Adjuster.role.yaml`). Endpoints, operations, and fields must be listed at both levels to be available to the call.
    c. The resource access is the intersection of the resources accessible to the service (as defined in the `serviceUser access.yaml`) and the resources available to the user (as defined in the `internal access.yaml` using the resource access ID of `aapplegate@acme.com`). In the base configuration, the `serviceuser access.yaml` files make all resources available. Therefore, logically speaking, the service-level resource access does not specify any restrictions. The call can access any resource provided it is available through the user-level resource access.
11. ClaimCenter provides the response to the initial call.

Services with user context: external users

In the following example, an API call is triggered by the Acme FNOLReporter service on behalf of Ray Newton, who is an external user.


0oaqt9pl1vZK1kybt0h7, aSecret,
scp.cc.acme_fnolreporter,
cc.service, cc.allowusercontext,
tenant.acme, project.default, planet_class.prod
1
service
4
2 authentication
Guidewire
Hub
3
JWT
Insurer
IdP
API request
with JWT
and
user context
header
API
response
11
Insurance
Suite
application
10
5
scp.cc.acme_
fnolreporter
acme_fnol
reporter.
role.yaml
role.
yaml
files
6
gwa.prod.cc.
Insured
Insured.
role.yaml
role.
yaml
files
7
cc.service
serviceUser
access.yaml*
access.
yaml
files
8
cc_policy
Numbers
policyNumbers
access.yaml*
access.
yaml
files
9
cc_policy
Numbers
ext
user
RestAuth
Source
Creator
plugin
JWT includes:
"sub": "0oaqt9pl1vZK1kybt0h7",
"cid": "0oaqt9pl1vZK1kybt0h7",
"scp": [
"scp.cc.acme_fnolreporter",
"cc.service",
"cc.allowusercontext",
"tenant.acme",
"project.default",
"planet_class.prod",
]
User context header includes:
"sub": "rnewton@email.com",
"groups": [
"gwa.prod.cc.Insured"
],
"cc_policyNumbers": [
"PA-123456"
]
Process request
session user: extuser
endpoint access: intersection of acmeFNOLreporter.role.yaml and
Insured.role.yaml
resource access: intersection of serviceuser.access.yaml and
policyNumbers.access.yaml, using PA-123456

1. When FNOLReporter triggers an API call, it must first request a JWT from Guidewire Hub. The request for the JWT includes the client ID (`0oaqt9pl1vZK1kybt0h7`), the secret (`aSecret`), the application's API role (`scp.cc.acme_fnolreporter`), the application's resource access strategy (`cc.service`), the fact that the call will be made for a user (`cc.allowusercontext`) and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. Guidewire Hub authenticates the services based on the client ID and secret. It also verifies that the API role and resource access strategy provided in the request match what was specified when the service was registered with Guidewire Hub.
3. Guidewire Hub generates a JWT and sends it to the service. This JWT includes the client ID (`cid`) and a `scp` token claim which names the API role (`scp.cc.acme_fnolreporter`), the resource access strategy (`cc.service`), the `cc.allowusercontext` value (which indicates that user information is specified in an additional user context header), and additional deployment information.
4. The service sends the API request to ClaimCenter along with the JWT and a user context header that identifies the user (`rnewton@email.com`), the user's resource access strategy (`cc_policyNumbers`), and resource access ID (`PA-123456`).
5. ClaimCenter must determine the endpoint access at both the service level and the user level. It starts at the service level. Based on the API role value in the JWT (`scp.cc.acme_fnolreporter`), the `acme_fnolreporter.role.yaml` API role file is used to define the service-level access.
6. Next, ClaimCenter determines the user-level endpoint access. Based on the contents of the `groups` token claim in the user context header (`gwa.prod.cc.Insured`), the `Insured.role.yaml` API role file is used to define the user-level access.
7. ClaimCenter must also determine the resource access at the service level and the user level. It starts with the service-level resource access strategy. Based on the resource access strategy value in the JWT (`cc.service`), ClaimCenter grants service-level resource access as defined in the `serviceUser access.yaml` files. (* ClaimCenter starts with `serviceUser_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`serviceUser`".)
8. Next, ClaimCenter determines the user-level resource access strategy. Based on the resource access strategy value in the user context header (`cc_policyNumbers`), it grants user-level resource access as defined in the `policyNumbers access.yaml` files. (* ClaimCenter starts with `policyNumbers_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`policyNumbers`".)
9. To determine which proxy user to assign to the session, ClaimCenter calls the `RestAuthenticationSourceCreator` plugin. The user context header specified a resource access strategy of `cc_policyNumbers`. So, the plugin returns the proxy user for external users: `extuser`.
10. ClaimCenter processes the request.
    a. The session user is the proxy external user: `extuser`.
    b. The endpoint access is the intersection of the endpoints and operations defined granted at the service level (`acme_fnolreporter.role.yaml`) and at the user level (`Insured.role.yaml`). Endpoints, operations, and fields must be listed at both levels to be available to the call.
    c. The resource access is the intersection of the resources accessible to the service (as defined in the `serviceUser access.yaml`) and the resources available to the user (as defined in the `policyNumbers access.yaml` using the resource access ID of `PA-123456`). In the base configuration, the `serviceUser access.yaml` files make all resources available. Therefore, logically speaking, the service-level resource access does not specify any restrictions. The call can access any resource provided it is available through the user-level resource access.
11. ClaimCenter provides the response to the initial call.

Implementation checklist for services with user context

To configure the system APIs for authentication for services with user context, you may need to do the following tasks:

Task	More Information
Enable asymmetric encryption	"Enabling bearer token authentication" on page 83

Task	More Information
Provide deployment information	“Enabling bearer token authentication” on page 83
Register the caller application with Guidewire Hub	“Enabling bearer token authentication” on page 83
Create or modify API roles	“Endpoint access” on page 87
Review the resource access strategies provided in the base configuration	“Resource access” on page 95
Configure the proxy user	“Proxy user access” on page 101

Sending authenticated calls for services with user context

When a caller application wants to make a system API call for a service with user context, the caller application must:

1. Request a JWT from Guidewire Hub
2. Include the JWT (and the `GW-User-Context` header) with the system API call

Requesting codes and JWTs from Guidewire Hub

For more information on how to request JWTs from Guidewire Hub, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

Including JWTs with API calls

Once a JWT has been received from Guidewire Hub, it must be sent to ClaimCenter in the request object's Authorization header. The header must use this format:

```
Authorization: Bearer <token>
```

Note: If a call includes a JWT with the `cc.allowusercontext` token claim, but the request object's header does not contain a user context header, Cloud API treats the call as if it were coming from a standalone service. In other words, the call will be restricted to the access provided to the service. No user-based restrictions are applied because there was no user context header specifying a user.

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

chapter 8

OAuth2 client credential flow: Services with service account mapping

A *service* is an application that typically executes action without human intervention. Services typically have no user interface. Examples of services include:

- An FNOL application that takes a collection of First Notice of Loss information stored in a database or file and sends it to ClaimCenter so that claims can be created.
- A Metro Report application that provides ClaimCenter with information about police reports related to auto accidents.
- An application that uploads pictures of a covered location or vehicle, either when a policy is bound or after a loss has occurred.

This topic discusses how to execute authentication for services with service account mapping.

Authentication options for services

There are several ways a service can execute authentication with Cloud API.

Standalone service

A service can authenticate **as a standalone service**. In this case, the service executes the call as itself. It does not execute the call as a specific person or on behalf of a specific person. The service does not execute the call using a service account stored in ClaimCenter.

ClaimCenter designates a single internal user as the "proxy service user" for all standalone service calls. This proxy service user is attached to the standalone service session. If the call creates or modifies an object, the proxy service user is recorded as the user of record.

The primary advantage to this approach is that you need to manage authentication and authorization information at the service level only. There is no need to create and manage user accounts, user permissions, or additional mappings.

The primary disadvantage is that all standalone service calls share a single proxy service user. When a standalone service call creates or modifies an object, it may not be possible to identify which service made the call.

Service with user context

A service can authenticate **with user context**. In this case, the service presents information about itself. The call also includes a `GW-User-Context` header that provides information about a specific user. The user does not necessarily exist

in the ClaimCenter database. The call is able to do only the things that both the service by itself could do and the user by itself could do.

The specified user can be an internal user (a user who is listed in the ClaimCenter database). When this is the case, this internal user is attached to the session. If the call creates or modifies an object, this internal user is recorded as the user of record.

The specified user can be an external user (a user who is not listed in the ClaimCenter database). ClaimCenter designates a single internal user as the "proxy external user" for all service with user context calls that reference external users. When the specified user is an external user, the external proxy user is attached to the session. If the call creates or modifies an object, the external proxy user is recorded as the user of record.

The primary advantage to this approach is that a single service can send calls on behalf of different users. At the service level, you can specify service-level access. But, you can also further control access for each associated user.

There are two primary disadvantages. First, you must maintain access information at two levels: the service level and at the user level. Second, a service can specify any user in its header. There is no way to restrict a given set of users for use by a given service.

Service with service account mapping

A service can authenticate **with service account mapping**. In this case, the service is automatically mapped to a "service account". The mapping information is specified elsewhere in the environment. The service account is a user account in the ClaimCenter database that is intended to be used only by the service and not by any person.

The primary advantage to this approach is that permissions and auditing for the call is tied to a service account that is listed in the ClaimCenter database. You can create a different service account for each service and have fine control over the permissions available to each service.

The primary disadvantage is that you must create and maintain service accounts in ClaimCenter for calls made by external applications. Also, you must maintain the mapping information that maps each service to its service account.

Comparing the different approaches

The following table compares each approach.

	Standalone service	Service with user context	Service with service account mapping
Does the call provide information about the service?	Yes, in the JWT.	Yes, in the JWT.	Yes, in the JWT.
Does there need to be a user account in the ClaimCenter database for the call?	No	If the associated user is an internal user, yes. If the associated user is an external user, no.	Yes. (This user account is the "service account".)
Does the call include information about a user or user account?	No	Yes, in the `GW-User-Context` header.	No. The call provides a client ID for the service, but the mapping of client ID to service account is stored elsewhere.
Which endpoints can the call access?	The endpoints available to the service's API roles	The endpoints available to both the service's API roles and the user's API roles.	The endpoints available to the service account.
Which resources can the call access?	All resources (in the base configuration).	The resources available to both the service and the user.	The resources available to the service account.
What is the session user set to?	The proxy service user.	If the associated user is an internal user, the internal user. If the associated user is an external user, the proxy external user.	The service account.

This topic focuses on authentication for services with service account mapping.

- For more information on authentication for standalone services, see “OAuth2 client credential flow: Standalone services” on page 47.
- For more information on authentication for services with user context, see “OAuth2 client credential flow: Services with user context” on page 55.

Overview of authentication for services with service account mapping

Authentication includes credentials and authorization. Authentication information for services is specified in JWTs, and information from these JWTs is recorded in the logs.

Credentials

When a service makes an API call, the service sends a client ID and secret to Guidewire Hub. Guidewire Hub authenticates the service by confirming that the client secret is correct. This is true for standalone services, services with user context, and services with service account mapping.

When a service authenticates with service account mapping, the service is mapped to a service account in the ClaimCenter database. However, there is no authentication at the service account level. Authentication occurs only at the service level.

For more information on how client IDs and secrets are registered with Guidewire Hub, see “Registering the caller application with Guidewire Hub” on page 85.

Authorization

Endpoint access for services with service account mapping

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

For a service-with-service-account-mapping call, the system APIs map the service to a service account in the ClaimCenter database. Then, ClaimCenter queries the operational database for this service account's user roles. The service is given endpoint access to all API roles whose names corresponds to the names of the service account's user roles.

For example, suppose that the ACME FNOL service is mapped to a service account named "acmeFNOL". The acmeFNOL account has two user roles: "ACME Adjuster" and "ACME Reinsurance Manager". The ACME FNOL service triggers a system API call. ClaimCenter maps the service to the acmeFNOL account and queries the database for the service account's user roles. Two user roles are returned: "ACME Adjuster" and "ACME Reinsurance Manager". ClaimCenter then gives the service the endpoint access defined in the API roles named "ACME Adjuster" and "ACME Reinsurance Manager".

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access for services with service account mapping

Resource access defines, for a given type of resource, which instances of that resource the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and service vendors. All of these callers can use the endpoint to access resources whose type is claim, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

Resource access is controlled by two features: resource access IDs and resource access strategies.

A *resource access ID* is a string that defines who the caller is or what the caller owns. Resource access IDs are used to determine which resources an authenticated caller has access to through API calls.

- A resource access ID can identify *who the caller is*. For example, the resource access ID for a service provider is the provider's Address Book ID. Typically, a service provider can access all claims which include this ID in the list of associated contact IDs.
- A resource access ID can identify *what the caller owns*. For example, the resource access ID for a policyholder is a list of one or more policy numbers. Typically, a policyholder can access all policies with those policy numbers.

A resource access strategy is a set of logic that identifies the meaning of a resource access ID. The base configuration includes the following resource access strategies for service account:

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`cc_username`	Internal users and service accounts	A ClaimCenter account name	Any information this account could see in ClaimCenter based on their associated Access Control Lists (ACLs).

When a service makes a service-with-service-account-mapping call, the service is mapped to a service account name. This account name is used as the resource access ID, and the cc_username strategy is used. This strategy consists of system API logic that matches, as closely as possible, the user's access as defined in the base configuration's Access Control Lists (ACLs).

For more information on how resource access behaves, see “Resource access” on page 95.

Proxy user access for services with service account mapping

Proxy user access is not applicable for services with service account mapping. The service account is used as the user for the session. Therefore, even though the call is coming from a service, there is no need to assign a proxy user.

JWTs for services with service account mapping

JSON Web Tokens (JWTs) contain token claims. (In standard JWT parlance, these are referred to simply as "claims". To avoid confusion with claims in the property and casualty insurance sense, this documentation always refers to JWT claims as "token claims".) A *token claim* is a piece of information asserted about the bearer of the token, such as the bearer's name. For bearer token authentication, authentication information is stored in token claims.

When a service makes a call with service account mapping, only some of the information in the JWT is specific to Cloud API authentication. The following are the token claims in a JWT for a service-with-service-account-mapping call that pertain to Cloud API authentication.

```
"sub": "<clientId>",
"cid": "<clientId>"
```

- `sub` is the subject of the token. This is set to the service's client ID.
- `cid` is the client ID of the service. This is also set to the service's client ID.

For a service-with-service-account-mapping call, the JWT may contain a `scp` token claim. But, unlike calls from standalone services or services with user context, there is no authorization information specific to Cloud API in the `scp` token claim. All authorization information comes from the service account that the service is mapped to.

Mapping services to service accounts

Service account mapping

The *service account mapping* is a set of name/value pairs that map a service's client ID to the name of a user in the ClaimCenter database. This user is referred to as the *service account*.

When a service call is received, Cloud API checks to see if the client ID is listed in the service account mapping.

- If the client ID is found, the call is treated as a service-with-service-account-mapping call. The service account becomes the session user, and the access associated with the service account defines what the service can do.
- If the client ID is not found, the call is treated as either a service-with-user-context call or a standalone service call.

For example, suppose the following service account mapping exists.

Client ID	User Name
0oaqt9pl1vZK1kybt0h7	acmeDocuments
0oapqkzpmaHfIU0sI0h7	acmeCSRPortaleast
0oaer46gh823d777er0x	acmeCSRPortalwest

Suppose a call is received from a service with a client ID of "0oaqt9pl1vZK1kybt0h7". This client ID exists in the service account mapping. Therefore, the call is treated as a service-with-service-account mapping call. The call is associated with the service account whose user name is acmeDocuments.

Similarly, suppose a call is received from a service with a client ID of "0oa33344455566677788". This client ID does not exist in the service account mapping. Therefore, the call is treated as either a service-with-user-context call or a standalone service call.

Every service has a single client ID. Therefore, for a given service, either all calls are treated as service-with-service-account-mapping calls, or none of the calls are treated as service-with-service-account-mapping calls. If the client ID is in the service account mapping, it is the former. If not, it is the latter.

Storing service account mapping

Service account mapping can be stored in different locations. For every service call, Cloud API checks all locations. Cloud API uses the first mapping it finds. So, if a client ID is mapped in multiple places, only the first mapping is used. If a client ID is not listed in any of these locations, Cloud API treats the call as a either a service-with-user-context call or a standalone service call.

From a technical perspective, service account mappings can be spread out across all of these locations. However, insurers may find it easier to manage service account mappings if a single location is used.

Storing service account mapping: Guidewire Cloud Console

The first place Cloud API checks is the **Guidewire Cloud Console** (GCC) configuration variables. Insurers can make these changes on their own through the GCC user interface.

Service mapping entries in GCC configuration variables use the following syntax:

```
PLUGIN_AUTHENTICATIONVERIFIER_SUBJECTMAPPINGS_<sub>=<username>
```

where:

- *<sub>* is the value of the JWT's sub token claim (which is set to the client ID).
- *<username>* is the user name of the service account.

To deploy changes to GCC configuration variables, you must restart the server.

Storing service account mapping: config.properties

The next place Cloud API checks is in **the `config.properties` file**. Entries in this file use the following syntax:

```
plugin.PLUGIN_AUTHENTICATIONVERIFIER_SUBJECTMAPPINGS_<sub>=<username>
```

where:

- *`<sub>`* is the value of the JWT's sub token claim (which is set to the client ID).
- *`<username>`* is the user name of the service account.

To deploy changes to the `config.properties` file, you must restart the server.

Storing service account mapping is `config.properties` may be appropriate in development instances. But, Guidewire does not recommend storing service account mapping is `config.properties` in production instances.

The service account

A service account is intended to be used only by a single external service making Cloud API calls. Typically, the service account name reflects the service (such as `acmeDocuments`) and does not look like a person account name (such as `aapplegate`).

The service account needs to have the permissions appropriate for the service.

Guidewire recommends to not use service accounts for user activity. In other words, do not have users log on to ClaimCenter using a service account.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	The user name of the service account

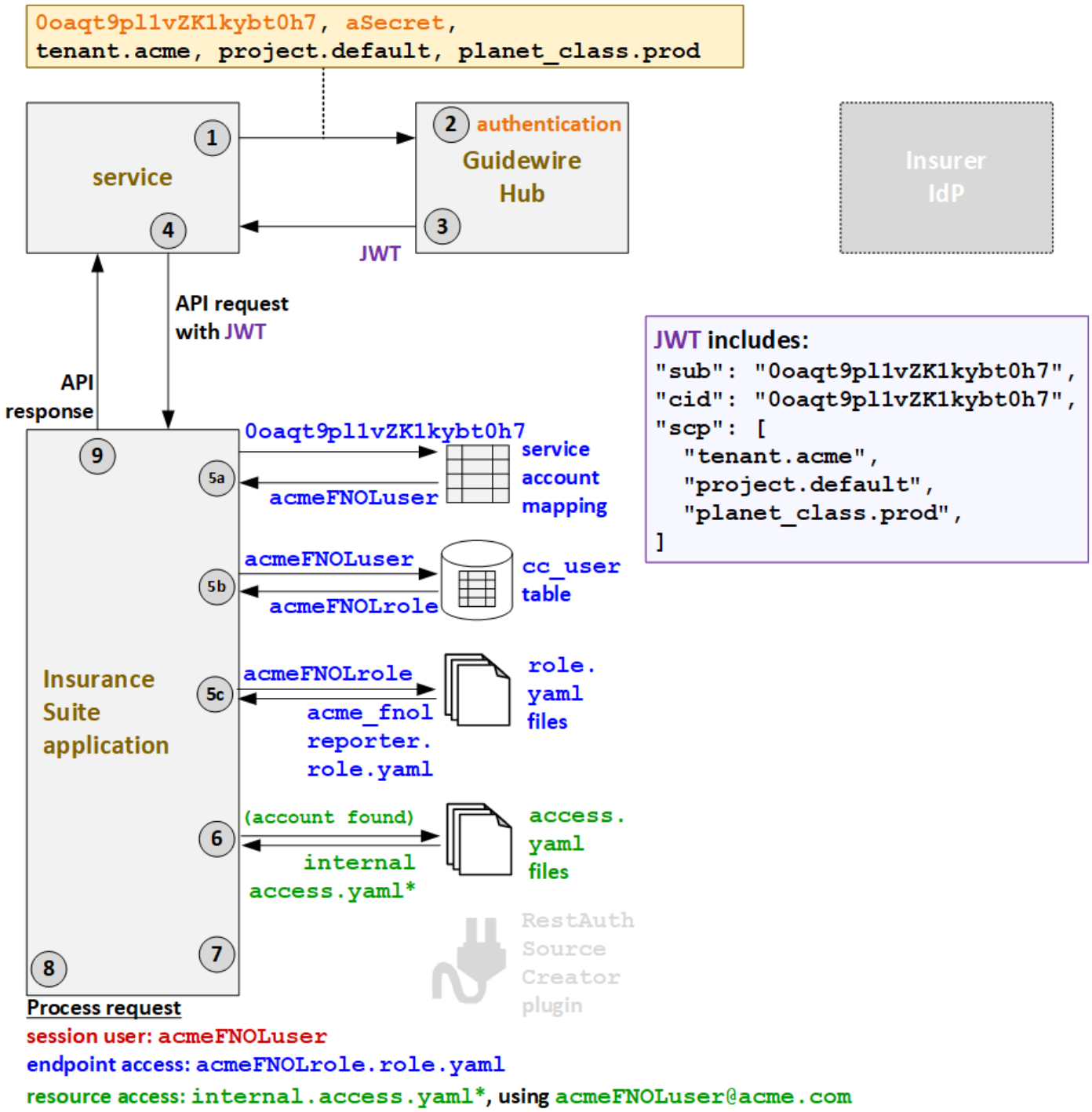
Example flow for services with service account mapping

The following diagram identifies the flow of authentication and authorization information for services with service account mapping. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by the Acme FNOLReporter.



1. When FNOLReporter triggers an API call, it must first request a JWT from Guidewire Hub. The request for the JWT includes the client ID (`0oaqt9pl1vZK1kybt0h7`), the secret (`aSecret`), and additional deployment information (`tenant.acme`, `project.default`, `planet_class.prod`).
2. Guidewire Hub authenticates the services based on the client ID and secret.
3. Guidewire Hub generates a JWT and sends it to the service. This JWT includes the client ID (`cid`) and additional deployment information.
4. The service sends the API request to ClaimCenter along with the JWT.
5. ClaimCenter determines the endpoint access.

   a. First, ClaimCenter executes a lookup to map the client ID (`0oaqt9pl1vZK1kybt0h7`) to a service account name (`acmeFNOLuser`).
   b. Then, ClaimCenter queries for the user roles that this account name has. One role is returned: `acme_fnolreporter`.
   c. Based on the returned role, the `acme_fnolreporter.role.yaml` API role file is used to define the endpoint access.
6. Next, ClaimCenter determines the resource access strategy. Based on the fact that the service account lookup found a valid service account, ClaimCenter grants resource access as defined in the `internal access.yaml` files. (* ClaimCenter starts with `internal_ext-1.0.access.yaml`, but this file references additional `access.yaml` files whose name starts with "`internal`".)
7. Proxy user access is not relevant for services with service account mapping.
8. ClaimCenter processes the request.
   a. The session user is the service account: `acmeFNOLuser`.
   b. The endpoint access is defined by `acme_fnolreporter.role.yaml`.
   c. The resource access is defined by `internal access.yaml` using the resource access ID of `acmeFNOLuser`.
9. ClaimCenter provides the response to the initial call.

Implementation checklist for services with service account mapping

To configure the system APIs for authentication for standalone services, you may need to do the following tasks:

Task	More Information
Enable asymmetric encryption	"Enabling bearer token authentication" on page 83
Provide deployment information	"Enabling bearer token authentication" on page 83
Register the caller application with Guidewire Hub	"Enabling bearer token authentication" on page 83
Create the service accounts	"Mapping services to service accounts" on page 71
Create the service account mappings	"Mapping services to service accounts" on page 71
Create or modify API roles	"Endpoint access" on page 87
Review the resource access strategies provided in the base configuration	"Resource access" on page 95

Sending authenticated calls for services with service account mapping

When a caller application wants to make a system API call for a service with service account mapping, the caller application must:

1. Request a JWT from Guidewire Hub
2. Include the JWT with the system API call

Requesting JWTs from Guidewire Hub

For more information on how to request JWTs from Guidewire Hub, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

Including JWTs with API calls

Once a JWT has been received from Guidewire Hub, it must be sent to ClaimCenter in the request object's Authorization header. The header must use this format:

```
Authorization: Bearer <token>
```

Authentication failure error messages

For endpoints that return elements, when a given resource exists but the user lacks authorization to access it, Cloud API throws the following user message. This is the same message that is returned when the resource does not exist.

```
"status": 404,
        "errorCode": "gw.api.rest.exceptions.NotFoundException",
        "userMessage": "No resource was found at path <path>"
```

For endpoints that return collections, Cloud API returns all resources that meet the criteria and for which the user has sufficient resource access. If a resource exists, but the user lacks sufficient authorization, Cloud API omits it from the results.

These approaches are considered to be more secure as they prevent malicious callers from being able to verify the existence of data that they are not authorized to access.

chapter 9

Unauthenticated callers

An *unauthenticated caller* is a user or service who provides no authentication information. Unauthenticated callers can access only metadata endpoints. Unauthenticated callers are typically callers who need information about the system APIs only.

This topic describes how to implement system API authentication for unauthenticated callers.

Overview of authentication for unauthenticated callers

Typically, authentication includes credentials and authorization. However, unauthenticated callers have no credentials and have a limited set of default authorization.

Credentials

By definition, an unauthenticated user has no credentials.

Authorization

Endpoint access for unauthenticated callers

Endpoint access defines the aspects of an endpoint's behaviors that are available to a caller. This includes:

- What endpoints and resource types are available to the caller?
- What operations can a caller call on the available endpoint?
- What fields can the caller specify in a request payload or get in a response payload?

Endpoint access is controlled by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers through API calls. API roles act as allowlists. By default, a caller has no endpoint access. When the caller is associated with one or more API roles, they gain access to the endpoints, operations, and fields allowlisted in each of those API roles.

When an unauthenticated caller makes a system API call, the system APIs automatically assign them the Unauthenticated role. In the base configuration, this role provides access to `openapi.json` endpoints only.

For more information on how API roles are configured, see “Endpoint access” on page 87.

Resource access for unauthenticated callers

Resource access defines, for a given type of resource, which instances of that resources the caller can access. For example, suppose there is a GET `/claims` endpoint that is available to policyholders, underwriters, adjusters, and

service vendors. All of these callers can use the endpoint to access resources whose type is `claim`, but none of the callers can access all of the claims. For example:

- A policyholder may be able to see only the claims associated with the policies they hold.
- An underwriter may be able to see only the claims for policies assigned to them.
- An adjuster may be able to see only the claims assigned to them.
- A service vendor may be able to see only the claims that have a service request assigned to them.

A *resource access strategy* is a set of logic that identifies the meaning of a resource access ID. Unauthenticated callers are automatically assigned the `default` resource access strategy. This strategy does not provide access to any business resources. It provides access to system API metadata only.

Strategy name	Persona using this strategy	The resource access ID is assumed to be...	Grants access to...
`default`	Callers who have presented no resource access strategy	Not applicable	Metadata resources only (such as openapi.json)

For more information on how resource access behaves, see "Resource access" on page 95.

Proxy user access for unauthenticated callers

When a caller makes a system API call, the internal ClaimCenter logic may trigger checks that are unrelated to endpoint access or resource access. For example:

- A caller may attempt to assign an activity to themselves. ClaimCenter must check to see if the caller has sufficient permission to own an activity.
- A caller may attempt to create a payment for $2000. ClaimCenter must check to see if the amount of the payment exceeds the caller's authority limit.

Unauthenticated users are not listed in the ClaimCenter operational database, and therefore do not have any system permissions or authority limits tied to them. In the unlikely case that an unauthenticated user triggers an internal check, the system APIs make use of proxy users. A *proxy user* is an internal user that is assigned to an external user or service when the API call is made. Whenever internal ClaimCenter logic must check to see if the caller has sufficient access, the proxy user is checked.

For more information on how proxy user access behaves, see "Proxy user access" on page 101.

JWTs for unauthenticated callers

An unauthenticated caller has no JWT. They are automatically assigned the Unauthenticated API role, and the `default` resource access strategy. In the base configuration, this provides access to API metadata only.

Logging

For each call, information about the caller is logged. The following table lists the fields that provide information about who the caller is, and where the logged value comes from.

Field	Value
`sub`	The value of the `sub` token claim from the JWT
`clientId`	The value of the `cid` token claim from the JWT
`user`	An empty string

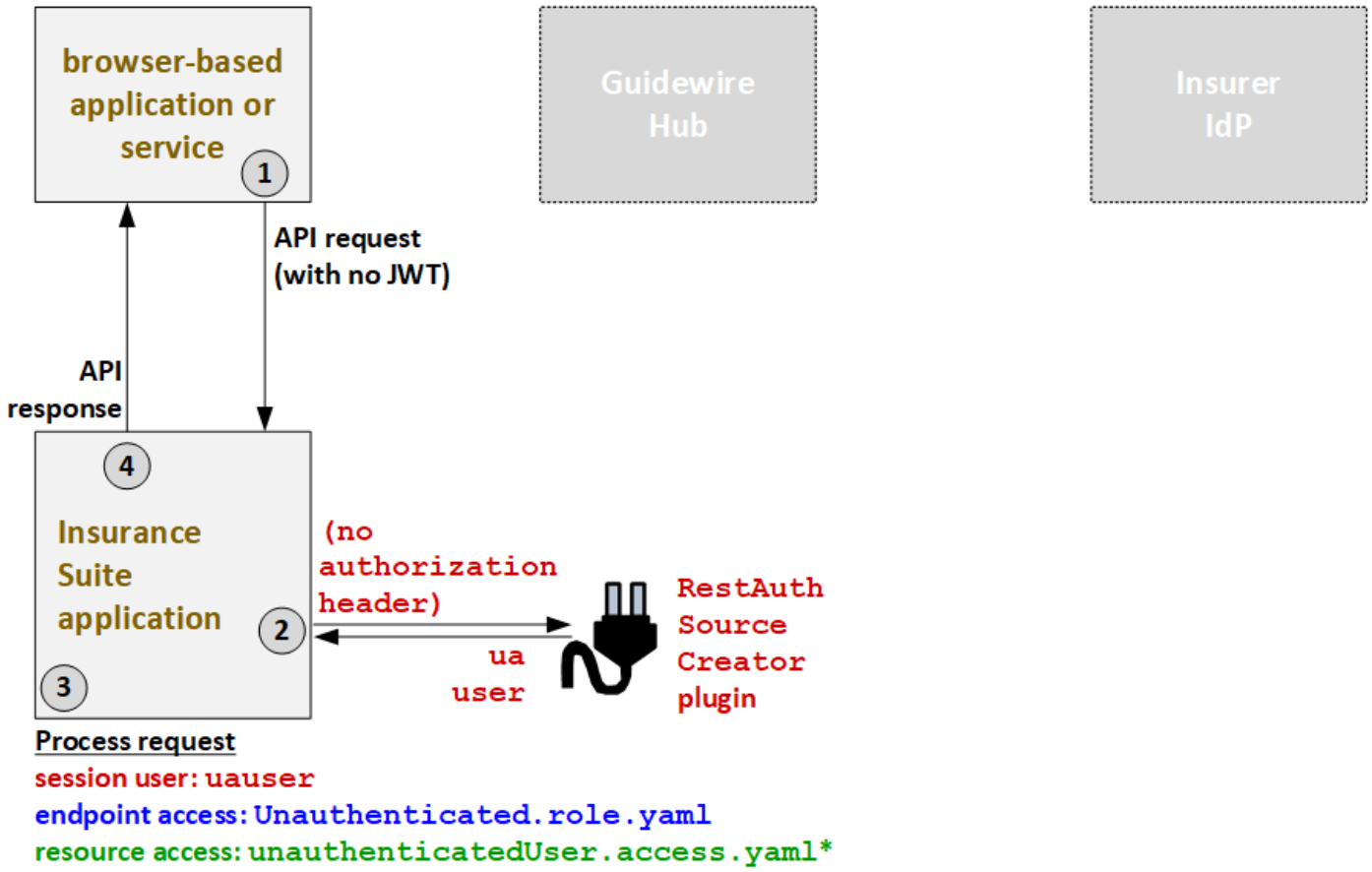
Example flow for unauthenticated callers

The following diagram identifies the flow of authentication and authorization information for unauthenticated callers. Colors are used in the following ways:

- Orange - credentials information
- Blue - endpoint access information
- Green - resource access information
- Red - proxy user and session user information

Some values are used to determine multiple types of access. These values initially appear as black (when they do not apply to a single type of access), and then later appear in one or more specific colors (to reflect the value is being used at that point in the process for a specific type of access).

In the following example, an API call is triggered by an unauthenticated caller.



1. The caller application sends the API request to ClaimCenter. The call includes no JWT, and no authentication information in the header.
2. Because the call has no authentication header, ClaimCenter grants endpoint access as defined in the `Unauthenticated.role.yaml` API role file. (This provides access to metadata endpoints only.)
3. Because the call has no authentication header, ClaimCenter grants resource access as defined in the `unauthenticatedUser.role.yaml` API role file. (This provides no access to business resources.)
4. To determine which proxy user to assign to the session, ClaimCenter calls the `RestAuthenticationSourceCreator` plugin. The call has no authentication header. So, the plugin returns the proxy user for unauthenticated users: `uauser`.
5. ClaimCenter processes the request.
   a. The session user is the proxy unauthenticated user: `uauser`.
   b. The endpoint access is defined by `Unauthenticated.role.yaml`.
   c. The resource access is defined by `unauthenticatedUser access.yaml`.
6. ClaimCenter provides the response to the initial call.

Implementation checklist for unauthenticated callers

To configure the system APIs for authentication for unauthenticated callers, you may need to:

1. Register the caller application with Guidewire Hub
2. Review and modify the Unauthenticated API role.
3. Configure the proxy user

To configure the system APIs for authentication for unauthenticated users, you may need to do the following tasks:

Task	More Information
Provide deployment information	"Enabling bearer token authentication" on page 83
Register the caller application with Guidewire Hub	*Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.
Review and modify the Unauthenticated API role	"Endpoint access" on page 87
Configure the proxy user	"Proxy user access" on page 101

To make a system API call for unauthenticated callers, the caller application does not need to request a code, request a JWT, or include a JWT with the system API call. The caller simply provides no authentication information.

part 3

Implementing authentication

This section provides information on how to execute each task for the implementation of authentication. This includes:

- How to enable bearer token authentication for a given instance of ClaimCenter
- How to configure endpoint access
- How to configure resource access
- How to configure proxy user access

chapter 10

Enabling bearer token authentication

Insurers must execute several steps to enable bearer token authentication for an instance of ClaimCenter.

1. ClaimCenter must be registered with Guidewire Hub.
2. ClaimCenter must be configured so that it can use asymmetric encryption.
3. ClaimCenter must have its deployment information specified.
4. The IdP must be configured to store and assert information about the users.
   - This is required for internal and external users only.
5. Every caller application must be registered with Guidewire Hub.

This topic details all of the tasks that fall into the category of enabling bearer token authentication.

Registering ClaimCenter with Guidewire Hub

Every cloud instance of ClaimCenter must be registered with Guidewire Hub. During the registration process, Guidewire provides you with an auth server URI for Guidewire Hub and a tenant ID for ClaimCenter.

For more information on how to register your instance of ClaimCenter with Guidewire Hub, talk to your Guidewire representative.

Enabling asymmetric encryption

Bearer token authentication for Cloud API uses *asymmetric encryption*. To verify a given JWT, ClaimCenter executes an *asymmetric public key lookup*. Periodically, ClaimCenter must request the keys used in these lookups from Guidewire Hub.

When you register ClaimCenter with Guidewire Hub, you are given an auth server URI and a tenant ID. For ClaimCenter to be able to request keys from Guidewire Hub, you must store the auth server URI and the tenant ID in certain plugin registries.

Enable asymmetric encryption

About this task

Before you can complete this task, you must register ClaimCenter with Guidewire Hub. You will need the `authServerUri` value provided at the end of registration. For more information, talk to your Guidewire representative.

The following steps identify how to complete this task in your instance of ClaimCenter. It may also be possible to complete this task by storing the `authServerUri` in Guidewire Cloud Property Services. For more information, talk to your Guidewire representative.

> **Note:** The auth server URI is used by the `SignatureKeyProviderPluginV1` plugin. In the base configuration, the plugin registry reads the value from the ClaimCenter `config.properties` file. Therefore, these instructions indicate how to modify the value in the properties file. If you have modified your configuration to read the value from other locations, then you will need to change the value in those locations as needed.

Procedure

Specify the auth server URI.

a) In Guidewire Studio, navigate to **configuration** > **config**, and open `config.properties`.

b) Add the following line to the file. (Note that this line may already be in the file as a comment. If so, you can simply uncomment the line.) `plugin.signaturekeyprovider.allowedissuers =`

c) Set the value of the `allowedissuers` properties to the value of the `authServerUri` provided to you by Guidewire.

Specifying deployment information

Checking deployment IDs

When processing an API call that uses bearer token authentication, Cloud API verifies that the tenant, project, and planet class specified in the JWT match the tenant, project, and planet class of this instance of ClaimCenter.

This check is accomplished using deployment IDs. A *deployment ID* is a string that specifies, for a given instance of ClaimCenter, the tenant, project, and planet class than the instance belongs to. Deployment IDs are formatted using *GRN* (Guidewire Resource Notation).

If you want bearer token authentication to work for a given instance of ClaimCenter, the instance's deployment ID must be specified prior to starting the instance. If the deployment ID is not specified, every call using bearer token authentication is returned with a null pointer exception.

For more information about specifying deployment information, see the *Guidewire Cloud Platform Documentation*.

Configuring the IdP

For internal users, the IdP must store:

- The user's credentials (for example, user name and password)

For external users, the IdP must store:

- The user's credentials (for example, user name and password)
- The list of API roles that are to be granted to the user
- The user's resource access IDs

The IdP must provide this information to Guidewire Hub when it asserts the user's identity. This information is used to verify the user's identity and to determine the user's endpoint and resource access.

Configure the IdP for bearer token authentication

Procedure

1. Configure your IdP so that it knows all of the API roles that are assigned to any external user.
   - Typically, this is done with IdP groups.
   - Each group name must be prefixed with "`gwa.<planetclass>.cc.`", where `<planetclass>` is set to either "`prod`", "`preprod`", or "`lower`".

- After this prefix, each group name must be identical to a system API role name.
- For example, to assign users to an API role named "Insured" for a production planet, the IdP group must be named "`gwa.prod.cc.Insured`".

2. Configure your IdP so that every user is associated with their user credentials (such as user name and password).
3. Configure your IdP so that every external user is associated with their API roles.
4. Configure your IdP so that every external user is associated with the correct resource access IDs:
   - For policy holders, this is an array of one or more policy numbers.
   - For service providers, this is a single Address Book unique identified (`gwabuid`).
5. Configure your IdP so that when a user is verified, the authorization information is asserted using the following attribute names:
   - User name is asserted as `cc_username`.
   - API roles are asserted as an array named groups.
   - Resource access IDs for policy holders are asserted as an array named `cc_policyNumbers`.
   - Resource access IDs for service providers are asserted as `cc_gwabuid`.

Registering the caller application with Guidewire Hub

Every caller application must be registered with Guidewire Hub. The information provided during the registration process varies based on whether the application is a browser-based application or a service application.

Register an application with Guidewire Hub

About this task

Before an application can request JWTs, the application must be registered with Guidewire Hub.

Procedure

1. Determine which auth flow the application will use. The auth flow must be one of the following:
   - Front end application (auth code flow for internal users and/or external users). If choosing this option, you must also specify whether you plan to use PKCE or a client secret.
   - Standalone service
   - Service with user context
   - Service with service account mapping
2. Contact Guidewire Cloud Operations and specify that you need an "InsuranceSuite REST API registration" using the desired auth flow.
3. Guidewire Cloud Operations sends a list of required information based on the selected auth flow. Provide this information to Guidewire Cloud Operations.
4. Guidewire Cloud Operations registers the application for OAuth based on the information provided. They will also send information to you that you need to further configure authentication, such as a client ID and client secret.

Results

Once you have the authorization information from Guidewire Cloud Operations, you can proceed with authentication configuration.

For further information on the difference between auth code flow with PKCE and auth code flow with client secret, refer to *Authentication with Guidewire Identity Federation Hub* in the *Guidewire Cloud Platform* documentation set.

chapter 11

Endpoint access

Endpoint access is defined by API roles. An *API role* is a list of endpoints, operations, and fields that are available to a set of callers when triggering system API calls. For example, API roles determine the following:

- What endpoints and resource types are available to the caller?
  - For example, can a given caller access the `/activities` endpoint?
- What operations can a caller call on the available endpoint?
  - For example, can a caller execute both a GET and a POST on the `/activities` endpoint?
- What fields can the caller specify in a request payload or get in a response payload?
  - For example, can a caller include the `priority` field in a POST `/activities` or retrieve the `assignedUser` in a GET `/activities`?

Note: The base configuration includes an internal super user whose user name is `su`. This user is not bound by endpoint access. Any authenticated call from `su` will have access to all endpoints.

API role files

API roles are defined in YAML files that are named *`RoleName`*`.role.yaml`. For example:

- `Insured.role.yaml`
- `ServiceRequestSpecialist.role.yaml`
- `Adjuster.role.yaml`

Location of API role files

API roles are declared in Studio in the **Integration** > **roles** directory. When checking for role files, the system APIs look in this directory only (and not in any subdirectories of the **roles** directory).

Note: In most cases when an insurer adds files to an instance of ClaimCenter, the best practice is to add these files to a subdirectory that is named after the insurer. However, an insurer cannot follow this practice with role files. For a role file to be available to the system APIs, the file must be declared directly in the **roles** directory. It cannot be declared in a subdirectory.

Structure of API role files

API roles are declared in Studio in the **Integration** > **roles** directory. Every file identifies:

- The role name
- The endpoints and endpoint operations for associated users

- The fields that associated users can view or edit

Note that these parts can be listed in any order.

API role names

Guidewire recommends using the same string of characters for the role name declared in the file and the role name as it appears in the file name. If an API role name must be multiple words, Guidewire recommends using underscores in the file name, and spaces in the file's `Role Name` section.

For example, if a new API role is for Fraud Investigators:

- Name the file `Fraud_Investigator.role.yaml`.
- In the file, declare the name as `name: "Fraud Investigator"`.

For API roles for internal users:

- There must be a user role in ClaimCenter assigned to the appropriate users.
- The API role name and the user role name must be the same.

For API roles for external users:

- The IdP must be able to associate each user with the API role name.
- The IdP must identify the role with a "`cc.`" or "`pc.`" substring followed by the role name.
  - The string in the IdP that identifies the role must start with `cc.`/`pc.`. The substring after the `cc.`/`pc.` must match the role name. For example, "cc.Manager" will be matched with the role named "Manager".)

For API roles for services:

- Guidewire recommends you name the role *insurercode_name* (such as `acme_locationphotos`), where:
  - *insurercode* is an insurer code, such as `acme`.
  - *name* is a meaningful name in lower case.
- When the service is registered with Guidewire Hub, the role must be named with an initial "`cc.`" or "`pc.`". But do not include the prefix in the API role file name or in the `Role Name` section.

API role endpoints

The `endpoints` section identifies the endpoints a grantee can use and the operations (GET, POST, PATCH, or DELETE) that a grantee can use on that endpoint. This section acts as an allowlist. By default, a caller cannot use any operation on any endpoint. To enable endpoint use, each endpoint and operation must be explicitly allowlisted.

The `endpoints` section contains a list of endpoints in the following pattern:

```
endpoints:
  - endpoint: <endpoint 1>
    methods:
    - <method 1 on endpoint 1>
    - <method 2 on endpoint 1>
  - endpoint: <endpoint 2>
    methods:
    - <method 1 on endpoint 2>
    - <method 2 on endpoint 2>
```

Wildcards in the endpoints section

You can use the asterisk (*) wildcard in the `endpoints` section.

A single * wildcard indicates access is provided for anything one level below the current endpoint level. For example:

- `/common/v1/activities/*` means "anything one level below `/activities`".
- `/common/v1/activities/*/notes` means "the notes for anything one level below `/activities`".

A double ** wildcard indicates access is provided for anything below the current level. For example:

- `/common/v1/activities/**` means "any resource or endpoint that can be accessed from the `/common/v1/activities path`".

Exercise caution when using **

Guidewire recommends that insurers exercise caution when using the ** wildcard. This is because later releases of the system APIs may add new endpoints that users will unexpectedly have access to through ** wildcards. For example, suppose in release 1.0 that an insurer creates an API role that provides access to `common/v1/activities/**`. As of release 1.0, this provides access to the following:

- `common/v1/activities/{activityId}`
- `common/v1/activities/{activityId}/assignees`
- `common/v1/activities/{activityId}/notes`

Then, in release 2.0, Guidewire adds the following endpoint:

- `common/v1/activities/{activityId}/confidentialAnalysis`

The API role will automatically have access to the new endpoint, even if this is not what the insurer intended when creating the API role for release 1.0.

API role accessible fields

The `accessibleFields` section identifies the fields of each resource that a grantee can view or edit. This section acts as a allowlist. By default, a caller cannot view or edit any fields on any resource. To enable viewing and editing, each resource, field, and permission must be explicitly allowlisted.

The `accessibleFields` section contains a list of resources in the following pattern:

```
accessibleFields:
  <Resource 1>:
    edit:
    - <fields the grantee can edit on resource 1>
    view:
    - <fields the grantee can view on resource 1>
  <Resource 2>:
    edit:
    - <fields the grantee can edit on resource 2>
    view:
    - <fields the grantee can view on resource 2>
```

Allowlisting resources

Resources can be named in several ways. You can name the resource explicitly. For example, the following specifies permissions for the Activity resource only:

```
accessibleFields:
  Activity:
    edit:
    - <fields the grantee can edit on this resource>
    view:
    - <fields the grantee can view on this resource>
```

You can also use the "*" wildcard. In this context, it means "all resources available to the endpoints listed in the endpoints section". For example, the following specifies permissions for all resources available to the role's endpoints:

```
accessibleFields:
  "*":
    edit:
    - <fields the grantee can edit on this resource>
    view:
    - <fields the grantee can view on this resource>
```

Allowlisting fields

For every resource, you can specify two field-level permissions: `edit` and `view`. If a permission is not explicitly listed, then callers will not have that permission for any fields on the resource.

Field-level permissions can be named in several ways. You can explicitly name the field and permission. For example, the following grants edit access to the `Activity` resource's `subject` field , and view access to priority `field` and the `subject` field.

```
accessibleFields:
  Activity:
    edit:
    - "subject"
    view:
    - "priority"
    - "subject"
```

You can also use the "*" wildcard. In this context, it means "all fields". For example, the following grants edit access to the `subject` field on the `Activity` resource, and view access to all fields.

```
accessibleFields:
  Activity:
    edit:
    - "subject"
    view:
    - "*"
```

API role example

This is the contents of the `Adjuster.role.yaml` file:

```
endpoints:
  - endpoint: "/admin/v1/openapi.json"
    methods:
    - "*"
- endpoint: "/claim/v1/**"
    methods:
    - "*"
- endpoint: "/common/v1/**"
    methods:
    - "*"
accessibleFields:
  "*":
    view:
      - "*"
    edit:
      - "*"
name: Adjuster
```

Note the following:

- A user with the Adjuster role has access to the following endpoints:
  - The Admin API's `openapi.json` endpoint
  - All endpoints in the Claim API.
  - All endpoints in the Common API.
- A user with the Adjuster role can use all operations (GET, POST, PATCH, and so on) on those endpoints.
- A user with the Adjuster role can view and edit all fields on the available endpoints.

Assigning API roles to callers

The manner in which API roles are assigned to a caller depends on the type of caller.

Assigning API roles to internal users

An *internal user* is a person who is listed as a user in the ClaimCenter operational database. For example, Andy Applegate, a ClaimCenter adjuster, is an internal user.

When an internal user makes a system API call (using either basic authentication or bearer token authentication), ClaimCenter queries the operational database for this internal user's user roles. The user is given endpoint access to all API roles whose names correspond to the names of the user's user roles.

For example, suppose that Andy Applegate is an internal user with two user roles: Adjuster and Reinsurance Manager. Andy Applegate triggers a system API call. When the API call is received, ClaimCenter queries the database for

Andy's user roles. Two user roles are returned: Adjuster and Reinsurance Manager. ClaimCenter then grants Andy the endpoint access defined in the API roles named "Adjuster" and "Reinsurance Manager".

API roles and ClaimCenter user roles

For internal users, there are two sets of roles that are used to enable endpoint access. For each logical role, there is a ClaimCenter user role and an API role with the same name. The API role provides endpoint access comparable to the user role.

The following table compares the two types of roles.

Type of role	What does the role specify?	For internal users logging directly in to ClaimCenter...	For internal users who trigger a system API call...	Where is the role configured?
InsuranceSuite user role	A set of system permissions	This specifies what the user can do through the ClaimCenter user interface	This is used to determine which API roles to assign to the user	The **Roles** screen on the ClaimCenter **Admin** tab
API role	A list of accessible endpoints, operations, and fields	Not applicable	This specifies the endpoint access provided to the user	A set of YAML files in Studio

Assigning API roles to external users and services

An *external user* is a person who is known to the insurer but who is not listed as a user in the ClaimCenter operational database. A *service* is a service-to-service application.

When external users or services make API calls, the call includes a JWT (JSON Web Token). This JWT contains authentication information about the caller, including the API roles to assign to the caller.

Parsing API role information in a JWT

When ClaimCenter receives a JWT, it looks for the API roles to grant. This information is either in the `groups` token claim (for external users) or the `scp` token claim (for services). Any value in the appropriate token claim is assumed to be an API role if it starts with "`gwa.<planetclass>.<xc>.`", where `<planetclass>` is set to either "`prod`", "`preprod`", or "`lower`", and where <*xc*> is the application code (such as "cc" or "pc"). For each value, ClaimCenter does the following:

1. It strips off the prefix "`gwa.<planetclass>.<xc>.`" substring.
2. It converts any blanks in the remaining to string to underscores.
3. It then searches for an API role file with the same name.

For example, suppose there is an external user JWT with a `groups` token claim that contains one string: "gwa.prod.cc.Customer Service Representative". ClaimCenter removes the initial "gwa.prod.cc." and converts the spaces to underscores, resulting in the string "Customer_Service_Representative". It then searches for an API role whose file name is "Customer_Service_Representative.role.yaml".

If there are no matches between the resulting strings and the API role names, the caller is given no endpoint access.

If there are multiple matches between the resulting substrings and API role names, the caller is given the union of the access specified in all matching roles. In other words, the API roles are ANDed together.

Assigning API roles to other types of callers

An *unauthenticated caller* is a user or service who provides no authentication information. Unauthenticated callers can access only metadata endpoints. Unauthenticated callers are automatically assigned the API role named `Unauthenticated`.

Reserved roles

Cloud API has two sets of reserved roles. These roles are used either by Cloud API or by other Guidewire features or services. Use caution when modifying these roles, as modifications may prevent the relevant Guidewire feature or service from behaving as expected.

Reserved roles for special types of Cloud API callers

Cloud API includes the following reserved roles for special types of callers:

- Unauthenticated.role.yaml - This role defines access for callers who provide no authentication information.

Reserved roles for other Guidewire services and features

Cloud API includes the following reserved roles for Guidewire services and features outside of Cloud API:

- gw_claimautomation.role.yaml - For calls made by the Claims Autopilot feature.
- gw_cloudrulesui.role.yaml - For calls made by the Cloud Rules UI service.

Designing API role files

API roles are typically designed either for multiple users or for a single service.

API roles designed for internal users and external users, such as Adjuster, Underwriter, Insured, or Account_Holder, are typically associated with multiple users. This is because there can be hundreds of users that need the same endpoint access. It is more efficient to design API roles that are reusable.

API roles designed for services are typically associated with a single service. This is because each instance of ClaimCenter interacts with a relatively small number of services and each service has its own access requirements. It is more efficient to create one role for each service rather than trying to define multiple, reusable roles.

Configuring API roles

You can modify the base configuration API role files, and you can create new ones.

To deploy changes to API files (either modifying an existing one or creating a new one):

- In development mode, you can hotswap the files.
- In production mode, you must restart ClaimCenter.

Create an API role file

Procedure

1. In Guidewire Studio, navigate to **configuration** > **config** > **Integration** > **roles**.
2. Right-click the **roles** folder, and then select **New** > **File**.
   - In order for the file to be available to the system APIs, the file must be declared directly in the **roles** folder. It cannot be declared in a subfolder.
3. In the **New File** dialog, enter the name of the role file. Name the file `*RoleName*.role.yaml`.
4. Specify the role name as: `name: *roleName*`.
5. Specify the endpoints, operations, and fields that the role grants. You can use the base configuration API role files as a reference.
6. Deploy the file by either hotswapping or restarting ClaimCenter.

Modify an API role file

Procedure

1. In Guidewire Studio, navigate to **configuration** > **config** > **Integration** > **roles** and open the file.
2. Modify the file as needed.
3. Deploy the file by either hotswapping or restarting ClaimCenter.

API roles and localization

If your instance of ClaimCenter uses one or more languages other than English, there are additional behaviors to be aware of.

Internal users and user role queries

When an internal user makes an API call, ClaimCenter queries the database for the user's user roles. This query returns the user roles using the ClaimCenter default application language, as specified by the `DefaultApplicationLanguage` parameter in `config.xml`. These results are then compared to the names of the API roles. Whenever there is a match, the internal user is given the access specified in the API role.

Therefore, if you change the `DefaultApplicationLanguage` parameter, the names of the user roles returned by the query will be in the new language. To ensure that internal users are granted the correct access, you must also change the names of the API role files used by internal users. Guidewire also recommends changing the name of the role within the file itself.

For example, suppose there is a user role in ClaimCenter named "Auditor". This user role maps to an API role named "Auditor.role.yaml". The ClaimCenter default language is changed to French. As a result of this change, the query now returns the role name as "Auditeur". To ensure that access to this role is granted appropriately, the API role file's name must be changed to "Auditeur.role.yaml".

External users and IdP roles

For external users, the roles associated with each user are stored in the IdP. When an external user makes a system API call, their associated roles are first stored inside a JWT. Each role is prefixed with a "cc." or "pc.". When ClaimCenter receives the call, it looks for any role names in the JWT prefixed with a "cc." or "pc." It strips of the prefix and then compares the remaining name with the names of the API roles. Whenever there is a match, the external user is given the access specified in the API role.

You can use any language for external roles, even if it is not the default language. But you must ensure that role names match between the IdP and ClaimCenter.

For example, suppose you wanted to create an external user role for accountants, and you wanted to do this using French. In ClaimCenter, the role could be named "comptable.role.yaml". The IdP would need to assert the appropriate users are associated with "cc.comptable" or "pc.comptable".

The prefix for external roles must always be "cc." or "pc.", even if the remainder of the role name uses a different character set, such as Japanese Kanji.

API roles for specific caller types

There are several roles that are designed for specific types of callers:

- All roles whose name is prefixed with "gw_"
- anonymous (used in PolicyCenter only)
- claimautomation_ext (used in ClaimCenter only)
- Unauthenticated

These roles are referenced by internal code or used by other Guidewire services and applications. Do not change the names for these role files, regardless of the language you are working in. Doing so will cause Cloud API authorization to not work properly.

chapter 12

Resource access

In order to view and edit information from ClaimCenter, a caller needs to be able to access one or more endpoints. This type of access is known as *endpoint access*. For example, if a caller has access to the GET `/claims` endpoint, that caller can view claims.

However, having access to a given endpoint does not mean a caller can view every resource that endpoint could return. In some cases, callers can access only certain instances of the relevant resource. For example, the GET `/claims` endpoint could be available to a policyholder, an adjuster, and a service vendor. But each of these users have access to a different set of claims:

- The policyholder can see only the claims associated with the policies they hold.
- The adjuster can see only the claims assigned to them.
- The service vendor can see only the claims that have a service request assigned to them.

This type of access is known as *resource access*. Resource access determines which instances of a given resource are available to a given caller. Resource access is defined by a set of resource access strategies. This topic describes how resource access strategies are assigned to a caller, how they are executed for each call, and how to interpret the base configuration files so that you can understand how resource access is executed.

> **Note:** If you need to configure the base configuration resource access behavior, consult your Guidewire account manager.

Resource access strategies

Strategies and IDs

A *resource access strategy* is a set of logic that identifies which resources a caller can access.

A *resource access ID* is a string that identifies either who the caller is or what the caller owns.

For each call, resource access is determined by executing the resource access strategy using the resource access ID as input. For example, suppose a given resource access strategy states "the caller can access information on claims related to the policies they own". And suppose, for a given call, the resource access ID is PA-123456. This would mean the following:

- The caller can access resources that are on claims related to policy PA-123456.
- The caller cannot access resources that are on claims related to policies other than PA-123456.

Some resource access strategies require a single resource access ID. Other resource access strategies allow for an array of resource access IDs.

The list of resource access strategies

The base configuration includes the following resource access strategies:

Strategy name	Persona using this strategy	Resource access ID is...	Grants access to...
cc_policyNumbers	Account holders and policy holders	An array of policy numbers	Resources associated with claims associated with any of the policies.
cc_gwabuid	Claims service provides	The ABUID (Address Book Unique IDentifier) of the service provider contact	Resources associated with any claims for which this user is an assigned service provider
cc_username	Internal users	A ClaimCenter user name	Resources this internal user could see in ClaimCenter based on their associated Access Control Lists (ACLs).
cc.service	Trusted service-to-service application	Not applicable	All resources
default	Callers who have been authenticated but specify no resource access strategy with the call	Not applicable	Metadata resources only (including API schemas and typelists)
unauthenticated	Callers who have not been authenticated	Not applicable	API schemas only

The strategy name is used in the JWT. It appears in the `scp` token claim to identify which resource access strategy to use for the call. Also, when appropriate, it appears as its own token claim to specify the resource access IDs.

For example, suppose that a given call is using the `cc_policyNumbers` resource access strategy with a resource access ID of PA-123456. The JWT would include the following.

```
"scp": [
  "cc_policyNumbers"
],
"cc_policyNumbers": [
  "PA-123456"
]
```

Determining a call's resource access strategy

Resource access strategies are assigned by internal code as described in the following table. For calls made by services with user context, two resource access strategies are assigned, one at the service level and one at the user level. For all other types of calls, only one resource strategy is assigned.

Strategy name	This is assigned to a call when...
cc_policyNumbers	Any of the following are true: • The JWT's `scp` token claim contains `cc_policyNumbers`, or • The call includes a user context header, and the header includes a `cc_username` token claim.
cc_gwabuid	Any of the following are true: • The JWT's `scp` token claim contains `cc_gwabuid`, or • The call includes a user context header, and the header includes a `cc_gwabuid` token claim.
cc_username	Any of the following are true: • The call is using basic authentication, or • The JWT's `scp` token claim contains `cc_username`, or • The call includes a user context header, and the header includes a `cc_username` token claim, or • The JWT specified a client ID that was mapped to a service account.
cc.service	The JWT's `scp` token claim contains `cc.service`.
default	The caller has been authenticated, but the JWT specifies no resource access strategy.

Strategy name	This is assigned to a call when...
unauthenticated	The caller has not been authenticated.

The cc.service resource access strategy

Most of the resource access strategies specify restrictions. These resource access strategies limit what a caller can view.

However, the `cc.service` resource access strategy specifies almost no restrictions. This is because this resource access strategy is designed to be used by services. Services are expected to be configured such that they access only the resources appropriate for the circumstance. Consequently, JWTs for API calls from services do not typically include resource access IDs.

Note that resource access for the different service-related auth flows behave as described here:

- For **standalone services**, calls use the `cc.service` resource access strategy. Therefore, they have unrestricted resource access.
- For **services with user context**, each call's resource access is the intersection of the service-level resource access and the user-level resource access. The service-level resource access is the `cc.service` resource access strategy, which has no restrictions. Therefore, logically speaking, a service-with-user-context call has resource access equivalent to the user-level resource access.
- For **services with service account mapping**, the service is mapped to an internal service account. The `cc.service` resource access strategy is not used. Rather, the call uses the `cc_username` resource access strategy.

Resource access files

Resource access strategies are defined in a set of resource access files. Each file is a YAML files whose name ends in `access.yaml`. For example, the `internal_core-1.0.access.yaml` file defines the base configuration resource access strategy for internal users. In Studio, access files are located in subdirectories of the `Integration/authorization` directory.

Types of access files

Broadly speaking, there are two types of access files: core and extension.

A *core access file* is an access file that defines one or more base configuration files. Core access files either have core in the name, or are located in a package with core in the path.

WARNING: Do not modify any core access files. Modifying these files can result in certain sets of callers being unable to execute API calls.

An *extension access file* is an access file that provides a location for extensions to base configuration resource access strategy behavior. Extension access files either have `ext` in the name, or are located in a package with `ext` in the path.

Note: If you need to configure the base configuration resource access behavior, Guidewire recommends that you consult your Guidewire account manager before attempting to modify any extension access files.

A strategy is defined across multiple files

Every resource access strategy is defined in multiple files. In the base configuration, all of the files for a given resource access strategy start with the same name.

For example, the `cc_policyNumbers` resource access strategy is defined in the following files:

- The `core.shared.v1` package's `policyNumbers_core-1.0.access.yaml`
- The `ext.shared.v1` package's `policyNumbers_ext-1.0.access.yaml` (which references `core.shared.v1.policyNumbers_core-1.0`)

Permissions and filters

Every access file contains a list of resources. Resources can be named in the singular (such as `Activity`) or in the plural (such as `Activities`).

When a resource is named in the **singular**, the information that follows applies to endpoints that return individual elements of that resource type. This includes endpoints whose operations are GET (for a single element), POST (for custom business actions), PATCH, and DELETE. For individual elements, there can be only one sections: `permissions`.

When a resource is named in the **plural**, the information that follows applies to endpoints that return collections of that resource type. This includes endpoints whose operations are GET (for a collection) and POST. For collections, there can be two sections: `permissions` and `filters`.

Resource permissions

Both individual element resources (such as `Activity`) and collection resources (such as `Activities`) can have a `permissions` section. This section defines the actions associated users can take on accessible resources.

The `permissions` section consists of a list of permissions, each of which is followed by a Boolean expression. The permission is granted if and only if the Boolean expression returns true.

For example, the following code defines the permissions for the `Claims` resource (for a collection of claims) as declared in the `policyNumbers_core-1.0access.yaml` file. The `view` permission is always granted. The `create` permission is granted if the Gosu expression `user.hasPolicyAccess(resource, Optional.of(data))` returns true.

```
Claims:
  permissions:

    view: true
    create: "user.hasPolicyAccess(resource, Optional.of(data))"
```

Permissions for element resources

For individual elements, you can specify `view`, `create`, `edit`, and `delete` permissions. You can also specify permissions for custom business actions. For example, if there is a POST `/activities/{activityId}/assign` endpoint, then for an `Activity` resource, you can specify an `assign` permission. For custom actions, the permission name must match the verb used at the end of the endpoint path.

For example, the following specifies permissions for the `Claim` entity. Note that is specifies standard `view` and `edit` permissions as well as a custom business action permission, `close`.

```
Claim:
        permissions:
        view: "perm.Claim.view(resource.Claim)"
        edit: "perm.Claim.edit(resource.Claim)"
        ...
        close: "perm.Claim.close(resource.Claim)"
        ...
```

WARNING: If you create a new endpoint that executes a custom action, do not name the endpoint with a name that would conflict with a base permission, such as `view`, `edit`, `create`, or `delete`. Doing so will result in unexpected permission behaviors.

If a given permission is not specified in an access file, then the permission defaults to the permission of the resource's parent. If a given resource does not have a permissions section, then all permissions default to the permission of the resource's parent.

Possible Boolean expressions

Any Gosu expression that returns true or false can be used as a permission's Boolean expression.

For permissions, the base configuration includes the following types of Boolean expressions:

- A Boolean value
- The keyword `__inherit` (in which case the permission is inherited from the resource's parent, such as `ClaimActivities...view: __inherit`)
- A Gosu expression, including:
  - A Gosu system perm expressions (such as `"perm.system.actview"`)
  - A Gosu resource perm expressions (such as `"perm.Activity.view(resource.Activity)"`)
  - A Gosu expression (such as `"!resource.Note.Confidential || resource.Note.Author == entity.User.util.CurrentUser || perm.Claim.viewconfidentialnotes(resource.Note.Claim)"`)
  - A Gosu method declared at the system API layer (such as `"gw.rest.core.pl.util.v1.ActivityInternalPermissionUtil.canApprove(resource.Activity)"`)

For more information on writing Gosu expressions that check for system permissions or resource permissions, refer to the *Rules Guide*.

In some cases, multiple expressions are listed on several lines, such as the following example. In this case, the expressions are ANDed together. All expressions must return true for the permission to be granted.

```
CheckSets:
  permissions:
    view:
      - __inherit
      - "perm.Claim.viewpayments(resource.Claim)"
```

Resource filters

Collection resources (such as `Activities`) can have a `filter` section. This section defines criteria the each member of the collection must meet in order to be accessible. They may be the only criteria that the resource must meet, or they may be in addition to other criteria added by ClaimCenter. (Filters are not used for individual element resources because you can use view and edit permissions to control the availability of an individual resource.)

The `filter` section consists of a list of one or more filter expressions. A filter expression can be:

- A Gosu expression that returns filter criteria, such as `gw.rest.core.pl.common.v1.activities.AssignedActivitiesFilter.`
- The keyword `__nofilter`, which indicates that there is no filter and all resources are accessible.

For example, the following code defines the filters for the `Claims` resource (for a collection of claims) as declared in the `policyNumbers_core-1.0access.yaml` file:

```
Claims:
  filter: gw.rest.core.cc.shared.v1.RestrictClaimsForPolicyNumbersFilter
```

Configuring resource access

Guidewire has designed the base configuration resource access framework to be appropriate, as much as possible, for every situation that an insurer may encounter.

If you believe that the base configuration framework does not meet your needs and you wish to configure resource access, Guidewire recommends that you consult your Guidewire account manager before attempting to modify any extension access files. This includes any of the following actions:

- Modifying the behavior of an existing `ext` access file
- Creating a custom system permission and referencing that permission in an access file
- Create a new Gosu permission method and referencing that method in an access file
- Create a new Gosu filter method and referencing that method in an access file

chapter 13

Proxy user access

A *proxy user* is an internal user account in the ClaimCenter database that is assigned to certain types of API calls made by external users or services. If the call records information or executes a permissions check that requires an internal user account, the proxy user account is used. This type of access is referred to as *proxy user access*.

Proxy user access is defined by the `RestAuthenticationSourceCreatorPlugin` plugin and a set of proxy users. This topic describes how to work with proxy users.

> **Note:** Proxy users do not apply to internal users (using either basic authentication or bearer token authentication). Proxy users are relevant only for external users, standalone services, services with external user context, and unauthenticated callers.

Proxy users

When a caller makes a Cloud API call, Cloud API checks to see if the caller has sufficient endpoint access and resource access. If they do, Cloud API hands processing over to the appropriate internal ClaimCenter logic.

The internal ClaimCenter logic may trigger code that can only be completed using a user account from the `cc_user` table. For example:

- The call may create or modify data. When this occurs, ClaimCenter records the name of the `CreateUser` or `UpdateUser`.
- The call may trigger a domain-level permission check.
  - For example, the call may attempt to assign an activity to the caller. To do this, ClaimCenter must verify that the caller has sufficient permission to own an activity.)
- The call may trigger an authority limit check.
  - For example, the call may attempt to create a payment for $2000. ClaimCenter must check to see if the amount of the payment exceeds the caller's authority limit.

When the caller is an internal user, ClaimCenter uses the internal user account for these types of code.

- The internal user is recorded as the `CreateUser` or `UpdateUser`.
- The internal user's user roles are checked for domain-level permissions as needed.
- The internal user's authority limit profiles are checked for authority limit checks as needed.

However, external users and services are not listed in the `cc_user` table. They cannot be recorded as a `CreateUser` or `UpdateUser`. They also have no system permissions or authority limits assigned to them. So, when a call is made by someone who is not an internal user, ClaimCenter assigns a *proxy user* to the call.

- If the call creates or modifies data, the proxy user is listed as the `CreateUser` or `UpdateUser`.

- If the call triggers a domain-level permissions check, the proxy user's user roles are checked.
- If the call triggers an authority profile check, the proxy user's authority profile limits are checked.

Types of proxy users

Fundamentally, there are three types of proxy users:

- The *external proxy user* is a proxy user assigned to calls made by external users and services with external user context.
- The *service proxy user* is a proxy user assigned to calls made by standalone services.
- The *unauthenticated proxy user* is a proxy user assigned to calls made by unauthenticated callers.

Note that for each type of caller listed above, there is only one proxy user. In other words, all external users and services with external user context make use of a single proxy user, the external proxy user. All standalone services make use of a single proxy user, the service proxy user.

Technically, there is a fourth type of proxy user, the *default proxy user*. This user is used in the unlikely situation that, for some reason, the regular proxy user cannot be assigned to the call.

Proxy users in the base configuration

The base configuration bootstrap data includes the following proxy users. (Bootstrap data is loaded when the product is installed. It is not a part of sample data.)

Proxy user type	Base configuration user	User role	Authority limit profile
External proxy user	`extuser`	External User	(none)
Service proxy user	`serviceuser`	Service User	Service User
Unauthenticated proxy user	`uauser`	Unauthenticated User	(none)
Default proxy user	`defaultuser`	Default User	(none)

To prevent anyone from logging in as one of these users, each of these users is created with a password that makes use of a character that is not valid Base64 encoding.

You can configure these users as needed. You can also create new users and designate any of them as a proxy user.

Proxy user assignment

Proxy users are assigned by the `RestAuthenticationSourceCreatorPlugin` plugin. The following table details the conditions that determine which proxy user to assign to a call.

Proxy user type	When this user type is assigned
External proxy user	The call includes the `cc_policyNumbers` scope or the `cc_gwabuid` scope
Service proxy user	The call includes the `cc.service` scope
Unauthenticated proxy user	The call has no authentication header
Default proxy user	The call requires a proxy user and for some reason no other proxy user could be assigned

The `RestAuthenticationSourceCreatorPlugin` plugin has four parameters, one for each proxy user type. Each parameter is set to the ID of an internal user. When the plugin must assign a given type of proxy user, the user with the associated user ID is assigned as the proxy user.

The following table lists the parameters, the parameter settings in the base configuration, and the users the settings correspond to.

Parameter	Base configuration value	Corresponding base configuration user
`externalUserPublicId`	`default_data:extuser`	`extuser`
`servicePublicId`	`default_data:serviceuser`	`serviceuser`
`unathenticatedUserPublicId`	`default_data:uauser`	`uauser`

Parameter	Base configuration value	Corresponding base configuration user
`defaultPublicId`	`default_data:defaultuser`	`defaultuser`

When is proxy user information used?

Proxy users as the "user of record"

Some of the actions that a user can execute in ClaimCenter require the user's name to be recorded in the database. For example:

- When an object (such as an activity or a note) is created, the user who created it is stored in the object's `CreateUser` field.
- When an object (such as an activity or a note) is modified, the user who modified it is stored in the object's `UpdateUser` field.

Actions that require a "user of record" can be triggered by system API calls. When the system API call is triggered by an internal user, the internal user is noted as the user of record. When the system API call is triggered by an external user or service, the proxy user is noted as the user of record.

Proxy users and system permissions

A system API call may trigger a check to see if the caller has a specific system permission. When this occurs, ClaimCenter

checks to see if the proxy user has a user role that includes the system permission.

- If the proxy user has the permission, processing continues as normal.
- If the proxy user does not have the permission, the action is prevented.

For more information on roles and permissions, refer to the *Application Guide*.

Proxy users and authority limits

A system API call may trigger a check to see if the caller has sufficient financial authority to execute a given action. When this occurs, ClaimCenter checks to see if the proxy user has an authority limit profile with the corresponding authority limit set to a sufficiently high amount.

- If the proxy user has a sufficiently high authority limit, processing continues as normal.
- If the proxy user does not have a sufficiently high authority limit, processing is suspended. ClaimCenter automatically creates an approval activity and assigns it to the appropriate user. If the activity is approved, processing for the underlying transaction continues.

For more information on authority limits, refer to the *Application Guide*.

Configuring proxy users

You can create new proxy users. Any user in the operational database can be a proxy user. There is no special quality that must be true about the user. The only conditions are:

- The user must have roles and authority limits appropriate to the users it will act as a proxy for.
- The user's ID must be referenced by the `RestAuthenticationSourceCreatorPlugin` plugin.
- Guidewire recommends that you prevent anyone from logging in as the user by giving the user a password that is not valid Base64 encoding, such as "…" (the ellipsis character).

Create a new proxy user

Procedure

1. From the ClaimCenter **Administration** tab, create a user role with the appropriate permissions.
   - If you want the proxy user to not be bound by system permissions, you can add all system permissions to this role.
   - For information on how to create user roles, refer to the *Application Guide*.
2. Create a user and assign that user to the user role.
   - For information on how to create users, refer to the *Application Guide*.
3. Determine the public ID of the new user.
   - You can do this by querying the database for the public ID. This can be done with a SQL query, or with a Gosu query executed from Studio.
4. In Studio, navigate to **plugins** > **registry** > **RestAuthenticationSourceCreatorPlugin**.
5. For each type of caller that the new proxy user applies to, set the parameter **Value** to the proxy user's public ID.

chapter 14

ContactManager authentication

Authentication for Cloud API for ContactManager is nearly identical to authentication for Cloud API for ClaimCenter. This topic identifies the differences between the two. If a topic is not explicitly discussed here, you can assume that it behaves in the same way for the two applications.

Supported caller types

Cloud API for ContactManager supports the following caller types:

- Basic auth
- Internal users (using bearer token auth)
- Standalone services
- Services with user context (where the user context references an internal user)
- Services with service account mapping
- Unauthenticated callers

Cloud API for ContactManager does not have any resource access files. Therefore, Cloud API for ContactManager does not support:

- External users
- Services with user context where the user named in the context is an external user

In PolicyCenter, there is a business need for a user to be able to create an account and quote a policy without being known to the insurer. Therefore, Cloud API for PolicyCenter supports anonymous users. The is no analogous business need for ContactManager. Therefore, Cloud API for ContactManager does not support anonymous users.

Resource access for ContactManager

The base configuration of Cloud API for ContactManager does not come with any resource access files. As a result, callers are not restricted by resource access.

Tag-based access to contacts

Contacts in ContactManager have contact tags. A *contact tag* is a tag that identifies one or more broad relationships that the contact has with the insurer.

The base application includes three tags:

- Client (a policy holder)
- Claim party (a contact involved in a claim)
- Vendor (a contact that provides service for a claim)

In order to view or edit a contact with a given type of tag, a user must have permissions for that action and for that tag. For example:

- In order to view contacts with the client tag, a user must have a "view client" permission.
- In order to edit contacts with the vendor tag, a user must have a "edit vendor" permission.

There are also two system permissions, `viewanytag` and `editanytag`, that let the associated users view or edit any contact, regardless of the contact's tags.

Base configuration behavior of tag-based permissions

In the ContactManager base configuration, all users have either the `viewanytag`, the `editanytag`, or both. This means that all base configuration user bypass tag-based permissions.

Cloud API access and tag-based permissions

Tag-based permissions apply to callers accessing contacts through Cloud API. In order to view or edit contacts with a given type of tag, the following must be true:

- For internal users and services with service account mapping:
  - Either the caller has an appropriate "view <tag>" permission and/or an appropriate "edit <tag>" permission, OR
  - The caller has the `viewanytag` permission and/or the `editanytag` permission
- For standalone services and services with user context:
  - Either the proxy user has an appropriate "view <tag>" permission and/or an appropriate "edit <tag>" permission, OR
  - The proxy user has the `viewanytag` permission and/or the `editanytag` permission

Note that for internal users and services with service account mapping, the permission check occurs against each caller individually. This means that, for a given "view <tag>" or "edit <tag>" scenario, some callers will be given access to the contact while others will not.

However, for standalone service, and services with user context, the permission check occurs against the one user designated as the proxy service user. This means that, for a given "view <tag>" or "edit <tag>" scenario, either all callers of that caller type will have access to the contact, or none of them will.