

CA API Gateway - 9.1

Thinking in Policy

Date: 02-Nov-2016



CA API Gateway - 9.1

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2016 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Table of Contents

Understanding Services and Policies on the Gateway	8
Gateway Environment and Configuration	10
CA API Gateway Design	12
CA API Gateway Components	13
Message Input	13
Message Processor and Policy Execution	13
Support Tooling	14
Developing a Policy	15
Policy Development is Iterative, with Environment Dependencies	15
Abstract the Policy into Modular Pieces	16
Local Environment Dependencies are Critical	16
Objects and Actions in Policy	16
Our Policy Data is Messages, Strings, Certificates, Dates, and More	17
Context Variables	17
Policy Elements are Program Actions	18
Composing Policy Logic	19
Composite Assertions	19
Other Composite Assertions	20
Building Common Structures	20
IF-THEN	20
IF-THEN-ELSE	21
Composite Assertions Ramifications	22
Complex Policy Structures	23

Policy Authoring Tips and Best Practices 25

Policy Performance Optimization 30

- Logging and Auditing 31
 - Auditing is Very Expensive 31
 - Logging is Cheaper, But Not Free 32
- Back-End Latency 32
 - Latency vs. TPS vs. Concurrency 32
 - Network Performance 33
- CPU-Intensive Operations 33
 - Per-Request SSL Session Initiation 33
 - Cryptographic Assertions 33
 - Create or Sign SAML Assertion 33
 - Issue JSON Web Token 34
- Data Transformation 34
 - Mode Switching 34
 - XSL Transforms 34
 - Regular Expressions 35
 - JSON Transform 35
- Caching 35
- Bottlenecks in Common Policy Elements 36
 - Database Operations 36
 - LDAP Operations 36
 - CA SSO and other SSO products 36

Turning Use Cases into Policies and Fragments 37

- Creating a Service and Service Policy 37
 - The Simple Case: Authentication and HTTP Routing Assertion 38
 - Gathering Credentials 38
 - Tokens 38
 - Validating Credentials/Authentication (AuthN) 38
 - Authorization (AuthZ) 39
 - LDAP 39
 - CA Single Sign-On and Other SSO Products 39
 - Routing 39
- Mediation Cases 39
 - Transport Mediation 40
 - Token Type Mediation 40
- Federation 40

Policy Fragments and Encapsulated Assertions	41
Auditing	41
Implementing a Policy Development Life Cycle	42
Terminology	42
Policies are Code and Deserve Similar Attention	43
Best Practices	43
GUI-Based Policy Migration Using the Policy Manager	44
Automated Policy Migration Using Management APIs, GMU, CMT	44
CMT/GMU	44
Conclusion	45

Thinking in Policy

The CA API Gateway product line focuses on the world of APIs and Services and is a part of your network and security infrastructure. The Gateway functions as a reverse proxy that deeply inspects network requests and responses from external client software, ranging from mobile applications to web browsers to business systems.

The CA API Gateway secures and protects your APIs and services from attacks, making them safe for public use. You can also use it to orchestrate related API calls into composite functions.



Tip: Unlike a workflow product or an ESB, the Gateway is not capable of very long lived workflow management and therefore is not suited to some pure ESB or Workflow use cases.

The following sections explain:

- How the product works and provides valuable information to prospective policy authors and architects.
- Core concepts of how the Gateway is configured and how these configuration items are used.
- How the policy language can address common use cases.
- Best practices for enterprise deployment to help you devise a strategy to move configuration between the corporate environments.

Understanding Services and Policies on the Gateway

At a high level, the basic unit of configuration in the CA API Gateway is the Service. For the Gateway, a Service is a logical construct that represents the sum of the API calls the client side can call to access the service that the Gateway is protecting.

Every service has a policy that implements an individual flow of data between the client and the back-end service. Policies may include other modular partial policies, known as [policy fragments](https://docops.ca.com/display/GATEWAY91/Policy+Fragments) (<https://docops.ca.com/display/GATEWAY91/Policy+Fragments>) or [encapsulated assertions](https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions) (<https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions>). Typically, modular policies have specific roles in authentication and authorization, routing to back-end services, and orchestration of larger functions.

In the Policy Manager, a policy includes assertions that determine the authentication method, identity credentials, transport method, and routing method for the web service or XML application. The specific types of assertions, their relative location, and the other assertions determine the properties and validity of a policy . During processing, the Gateway scans each policy assertion from top to bottom, assigning a 'succeed' or 'fail' outcome to each .

The following is the message processing model for a typical policy:

1. Service request arrives.
2. Request runs through the WS-Security processor, which does the following:
 - Decrypts encrypted sections and verifies WS-Security Signatures. The sign and/or encrypt order is chosen by the sender.
 - Optionally removes the default security header before routing.
3. Request runs through the policy assertions:
 - Routing assertion sends a request to the service server.
 - Remainder of policy assertions are applied to the service response.
4. Response run through the WS-Security decorator, which does the following:
 - Creates the default security header.
 - Applies the signatures specified by the policy.
 - Performs any encryption specified by the policy.
5. Response is returned to the client.

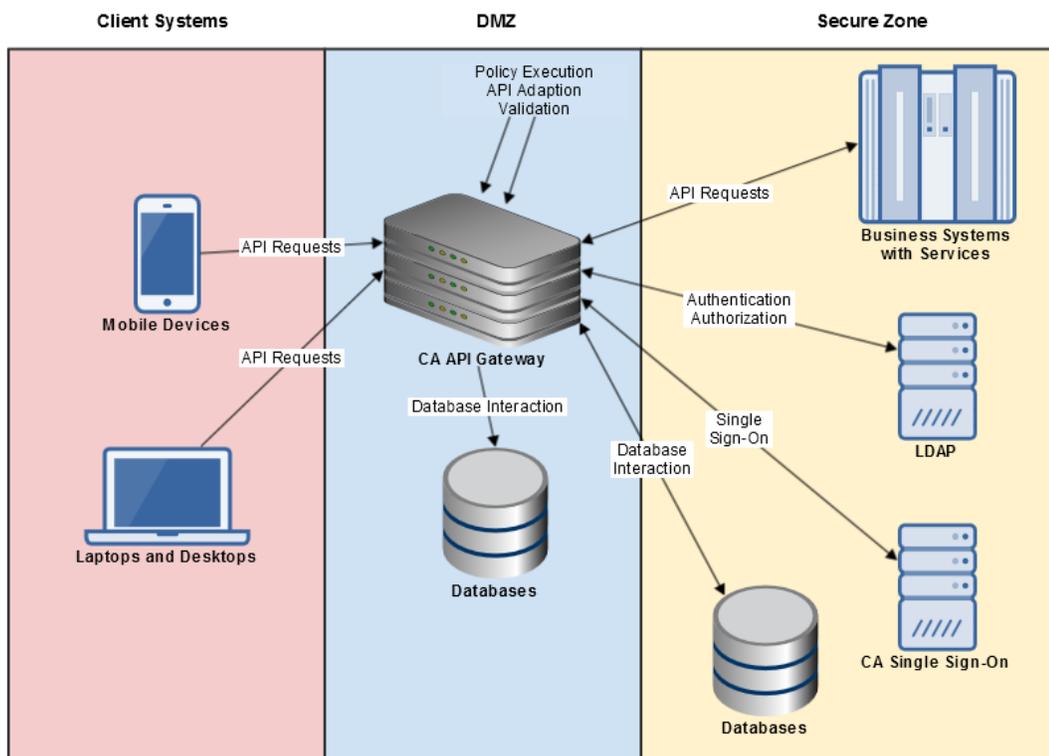
A Gateway policy can include configuration items derived from data often held in disparate groups within an enterprise. The security architects, the authorization and authentication group, the network infrastructure team, the application developers, and the service architects all serve roles in defining the details necessary for a policy. For example:

- The authorization and authentication team maintains the group membership strategy for the LDAP server.
- The network infrastructure team maintains the IP addresses of routers and load balancers.
- The service architects have access to the service URLs of back-end services, XML Schema documents, WADL & WSDL documents, etc.
- The security architects decide whether HTTP basic credentials over anonymous SSL are sufficient security, or if Mutually Authenticated SSL is necessary.

Gateway Environment and Configuration

Production Gateways are normally deployed in the DMZ and provide services for mobile and client systems located in less secure zones.

The business services being secured by the Gateway are normally in the secure zone. In the Network diagram below, there are multiple databases. The Gateway does not require any specific network location, but network and security architects often require that all databases are located in the secure zone.



Field implementations must cover both environment configuration and service "programming." Many individual policy operations require configuration data obtained by querying a database, an LDAP server, or an external system via SSL. Each has configuration, for example: credentials for the database or the SSL certificate for the external system. These parts are not directly in policy, but are in common configuration storage that must not be overlooked when migrating between environments.

Services and policies have "Environment Dependencies" specific to the physical environment the Gateway is deployed in; for example IP addresses of back-end systems and LDAP-specific details. It is critical to plan for these dependencies during policy development, especially for migration.

Specific environment specific values should be captured as [cluster properties](https://docops.ca.com/display/GATEWAY91/Cluster+Properties) (<https://docops.ca.com/display/GATEWAY91/Cluster+Properties>). These properties allow you to refer to common configuration items as symbols instead of literal values, making it very simple to make wholesale changes to many references. Cluster properties are referenced in the policy editor as "\${gateway.<property_name>}".

Example: In the [Route via HTTP\(S\) Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion), you use the cluster property `/${gateway.BusinessService}` in the host URL field, you can define the `BusinessService` cluster property as "dev.bizsvc.company.com" for your Development environment and as "qa.bizsvc.company.com (<http://qa.bizsvc.company.com/>)" in your QA environment. When you move a group of policies that all refer to that business service from Development to QA, you need change only the value of the cluster property. This concept can be use for most shared resources: app servers, database servers, LDAP services.

Most Gateway subsystems rely in various predefined cluster properties for details of configuration. A common example is the HTTP input subsystem, which uses the predefined cluster property `/${io.httpCoreConcurrency}` (<https://docops.ca.com/pages/viewpage.action?pageId=361529888>). This defines the number of simultaneous HTTP-based inbound connections allowed at one time. The default of 500 should be able to accommodate the traffic expectations for most common deployments.

All nodes in a Gateway cluster share the same configuration, greatly easing installation of larger environments. You create new cluster properties on demand, which provide tremendous power in implementing cluster-wide configuration changes. To learn more about the predefined cluster properties, see these topics:

- [Manage Cluster-Wide Properties \(https://docops.ca.com/display/GATEWAY91/Manage+Cluster-Wide+Properties\)](https://docops.ca.com/display/GATEWAY91/Manage+Cluster-Wide+Properties)
- [Cluster Properties \(https://docops.ca.com/display/GATEWAY91/Cluster+Properties\)](https://docops.ca.com/display/GATEWAY91/Cluster+Properties) (refer to all sub-topics)

CA API Gateway Design

The CA API Gateway consists of a protocol independent message processor, network input subsystems, and assertions that provide actions in the message processor, and output capabilities as well. The Gateway is not an app server.

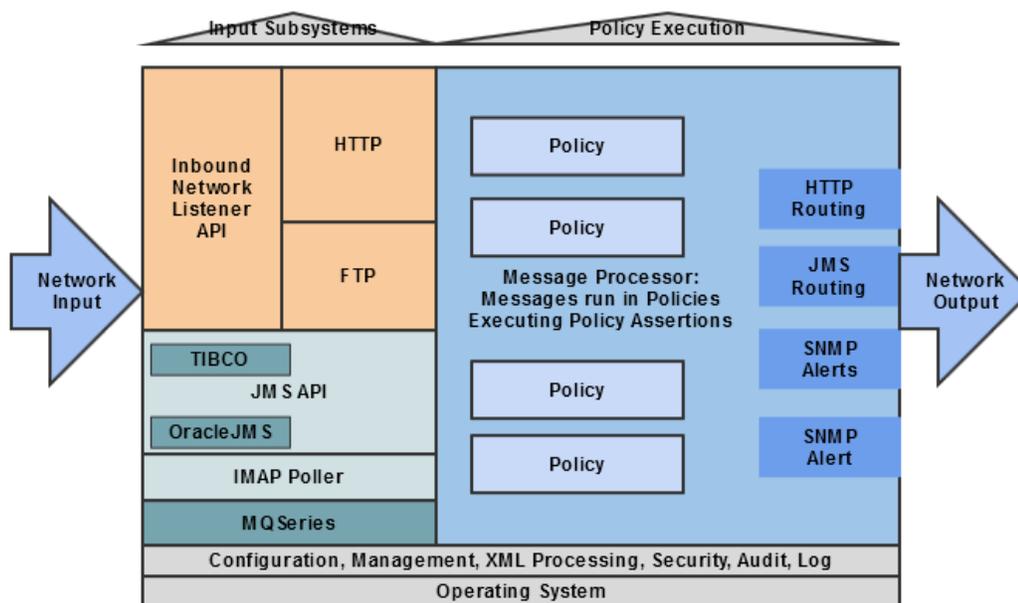
The message processor is protocol independent because it operates on a message--how that message is received (i.e., HTTP or FTP) is not considered. The back-end system that eventually receives the request may be of different technology. For example, it is common to submit a HTTP message a back end using a JMS Queue.

The product allows environment dependencies to have shared configuration between different services. For example, an environment may have multiple LDAP servers, with more than one Service that authenticates against each LDAP server. This configuration is supported, though it increases the chances of error.

The Gateway design is partially based on the concept of "XML Firewalling", so there are some similarities to an IP Firewall. By default, the Gateway assumes all messages are suspect and may be attempts to bypass security until proven otherwise. This is similar to best practices in IP firewalling, where all TCP ports are closed by default . As a result, the Gateway does not pass any data by default. If the inbound request contains anything unexpected, the Gateway is designed to block that information. The Gateway does not copy most HTTP headers from a request to the back end unless specifically configured to do so. All data must be explicitly referenced. Text data offers best performance, as interpreting binary data in a policy may require additional decoding and encoding steps.

CA API Gateway Components

The following diagram illustrates the major components of the CA API Gateway:



- [Message Input \(see page 13\)](#)
- [Message Processor and Policy Execution \(see page 13\)](#)
- [Support Tooling \(see page 14\)](#)

Message Input

Messages enter the message processor via network listeners (examples: HTTP, FTP) or via outbound connectors that initiate connections to external services (examples: IMAP Polling, Tibco EMS, MQ Series). The complete list of connectors varies between releases and there is an SDK to add additional network output protocols. Message input connectors are more complex, requiring a different, non-public SDK.

These input systems have two roles: Provide the network protocol as appropriate and prefill some network level details that are used later in policy. For example, the HTTP connector puts the remote IP address into a special context variable.

Message Processor and Policy Execution

The message processor is an efficient, multithreaded system that invokes functions (known as "assertions") on an inbound message as it follows policy logic.

Assertions are "aware" of their context within the currently executing policy. They get this information from a variety of sources, including the request and response messages, context variables, and the policy language.

Note that there is no specific message output system defined. This reflects how the Gateway is protocol independent. If a use case has a back-end system on JMS, then the policy uses the [Route via JMS Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+JMS+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+JMS+Assertion). If the back-end system is SFTP, then the policy uses the [Route via SSH2 Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+SSH2+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+SSH2+Assertion) (which provides SFTP routing). This underscores the important point: *Back-end requests are explicit policy actions.*

Support Tooling

The support tooling provides services used by multiple assertions and the listeners: Certificates, caching, connection pooling, audit, logging, management, token services, etc.

Developing a Policy

CA API Gateway policies provide APIs to client systems, while in turn utilizing back-end APIs. Enterprise needs governance similar to that used in software deployment to govern policy deployment.

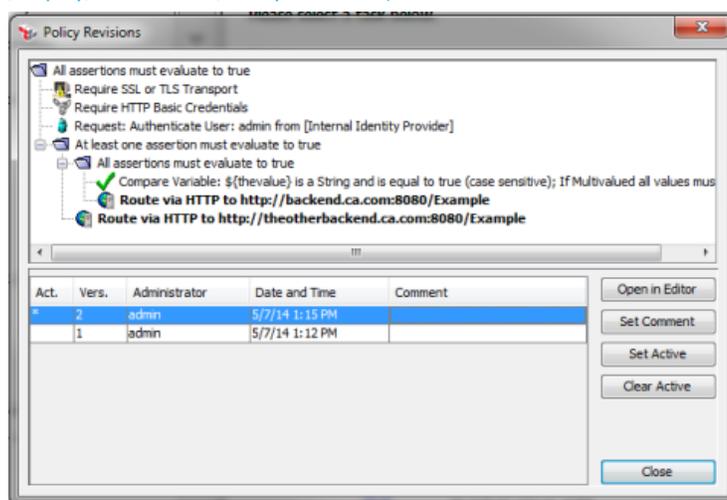
- [Policy Development is Iterative, with Environment Dependencies \(see page 15\)](#)
 - [Abstract the Policy into Modular Pieces \(see page 16\)](#)
 - [Local Environment Dependencies are Critical \(see page 16\)](#)
 - [Objects and Actions in Policy \(see page 16\)](#)
 - [Our Policy Data is Messages, Strings, Certificates, Dates, and More \(see page 17\)](#)
 - [Context Variables \(see page 17\)](#)
 - [Policy Elements are Program Actions \(see page 18\)](#)

Policies can dramatically alter the structure of API calls, so they must be deployed with care and rigor. Implement a "Policy Development Life Cycle" similar to standard software development life cycles.

Policy Development is Iterative, with Environment Dependencies

Fully planning a service or API deployment is not always practical as changes are invariably required. The revision control feature in the Gateway can help you manage such changes, but it is limited to a single environment.

The Policy Revision feature lets policy authors view and edit versions, or set a previously edited policy as the active version. For details on using this feature, see [Policy Revisions \(https://docops.ca.com/display/GATEWAY91/Policy+Revisions\)](https://docops.ca.com/display/GATEWAY91/Policy+Revisions).



To minimize policy changes, structure your policies to allow environmental dependencies to change without needing to update the policy (or create a new revision). The topic [Gateway Environment and Configuration \(see page 10\)](#) discusses how to use cluster properties to gain a more efficient workflow.

Abstract the Policy into Modular Pieces

Modular policies are known as *policy fragments* or *encapsulated assertions*. They are similar to the `#include` command in the C programming language: the fragment becomes part of the policy and executes using the same rules as the parent policy.

Modular policies allow subject matter experts and specialists to work on individual functions independently. This is similar to the standard practice of dividing large development projects into subsystems and assigning different teams to each subsystem.

For example, most organizations create a common "Authorization and Authentication" module as a policy fragment. Coupled with the built-in role-based access control, this can reduce errors dramatically: simply grant policy authors read-only access while allowing your security team read-write access. This prevents inadvertent changes by inexperienced staff.

For more information, see the following topics:

- [Policy Fragments \(https://docops.ca.com/display/GATEWAY91/Policy+Fragments\)](https://docops.ca.com/display/GATEWAY91/Policy+Fragments)
- [Encapsulated Assertions \(https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions\)](https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions)

Local Environment Dependencies are Critical

When policies move from a development to a test environment, most IT operations groups have different servers for various functions. This means different host names or IP addresses. You can map these dependencies to reduce effort and minimize errors.

For more information, see [Gateway Environment and Configuration \(see page 10\)](#).

Objects and Actions in Policy

The policy language uses high level objects that operate on actions. These actions are known as *assertions*. This originated from the WS-Policy working group, where the Assertion concept was central.

For more information, see [Policy Assertions \(https://docops.ca.com/display/GATEWAY83/Policy+Assertions\)](https://docops.ca.com/display/GATEWAY83/Policy+Assertions) and its subtopics.

Our Policy Data is Messages, Strings, Certificates, Dates, and More

There is a long list of types of data handled by the Gateway: Messages, Strings, Numbers, Certificates are some examples. Each assertion in the policy acts on objects based on configuration and some use configuration from environment dependencies.

Actions based on configuration might be a regular expression configured entirely by the text in the dialog. An example of actions based on configuration might be how the [Authenticate Against Identity Provider Assertion](https://docops.ca.com/display/GATEWAY91/Authenticate+Against+Identify+Provider+Assertion) (<https://docops.ca.com/display/GATEWAY91/Authenticate+Against+Identify+Provider+Assertion>) the configured LDAP server to verify the supplied credentials.

Context Variables

The Gateway policy language has access to hundreds of predefined context variables, which make data available in manageable units. For example, there are many variables available that return information about the target message. A great deal of interaction with the common assertions is via the default data that are always present during policy operation. You can refer to attributes of that data item via a variable reference such as `request.tcp.remoteip`.

In addition to the large library of predefined variables, policies can create new variables during policy execution. Many assertions automatically create their own context variables. For example, any of the XPath assertion (example: [Evaluate Request XPath Assertion](https://docops.ca.com/display/GATEWAY91/Evaluate+Request+XPath+Assertion) (<https://docops.ca.com/display/GATEWAY91/Evaluate+Request+XPath+Assertion>)) saves comprehensive data about the XPath in a series of context variables.

The 'Message' variable type is particularly complex. This variable type embodies the Gateway's ability to:

- consult multiple sources of data synchronously and asynchronously
- consult external systems to make decisions within policy
- perform all this without disturbing the flow of the main request and response data.

For information about specific context variables created by assertions, refer to the topic for each assertion. For example, the [Evaluate Request XPath Assertion](https://docops.ca.com/display/GATEWAY91/Evaluate+Request+XPath+Assertion) (<https://docops.ca.com/display/GATEWAY91/Evaluate+Request+XPath+Assertion>) topic lists the six `requestXPath.*` variables created by the assertion.

Policy Elements are Program Actions

Assertions act on the policy data mentioned previously. An assertion might change the request as configured in that specific policy. For example, the [Apply XSL Transformation Assertion \(https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion\)](https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion) could modify the request message according to the specific configuration in that specific policy.

All assertions return 'true' or 'false', indicating success or failure. The policy logic you implement will branch based on these results.

Composing Policy Logic

The Policy Manager uses a simple programming-like policy language that can create complex service policies. The *composite assertions* are the main entities—these let you create "AND" and "OR" constructs, using a folder metaphor that conveniently groups the relevant assertions and helps reinforce a parent/child relationship. Combine this ability with the Boolean "true/false" values returned by assertions and you can express powerful decisions.

- [Composite Assertions \(see page 19\)](#)
 - [Other Composite Assertions \(see page 20\)](#)
- [Building Common Structures \(see page 20\)](#)
 - [IF-THEN \(see page 20\)](#)
 - [IF-THEN-ELSE \(see page 21\)](#)
- [Composite Assertions Ramifications \(see page 22\)](#)
- [Complex Policy Structures \(see page 23\)](#)

Composite Assertions

The two main composite assertions are:

- [All Assertions Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion): All child assertions must resolve to 'true' for this assertion to succeed. This assertion operates similarly to logical 'AND'.
- [At Least One Assertion Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion): Only one child assertion needs to be true for this assertion to succeed. This assertion operates similarly to logical 'OR'.



Tip: For simplicity, the [All Assertions Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion) will be referred to as the "AND assertion", while the [At Least One Assertion Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion) will be the "OR assertion".

One very important point to note: these composites run their child assertions only until a state change:

- The "All assertions..." runs its child assertions sequentially until one fails.
- The "At least..." runs its assertions sequentially until one succeeds.

The ordering of the child assertions matter greatly, because all assertions after the state change are *not* executed.

Best practice tips: Optimize the order of assertions in your policy. Since policy execution for a branch halts upon failure, put your "cheapest" or most "lightweight" assertions earlier in your policy. For example: If SSL is always required, check for it early on in your policy. Checking for SSL is very fast and it quickly discards attack attempt with non-SSL messages.

"Quick" assertions are those that simply check for the existence of certain data; examples include:

- [Require SSL or TLS Transport Assertion \(https://docops.ca.com/display/GATEWAY91/Require+SSL+or+TLS+Transport+Assertion\)](https://docops.ca.com/display/GATEWAY91/Require+SSL+or+TLS+Transport+Assertion) checks whether SSL is present
- [Require HTTP Basic Credential Assertion \(https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion\)](https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion) checks for basic HTTP authentication
- [Require Timestamp Assertion \(https://docops.ca.com/display/GATEWAY91/Require+Timestamp+Assertion\)](https://docops.ca.com/display/GATEWAY91/Require+Timestamp+Assertion) ensures that there is a timestamp in the target message

By comparison, a more complex assertion like the [Apply XSL Transformation Assertion \(https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion\)](https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion) is considered "expensive" because it places greater demands on the CPU and the demand increases as the message gets larger.

Other Composite Assertions

In addition to the two main assertions listed above, these composite assertions are more esoteric variations:

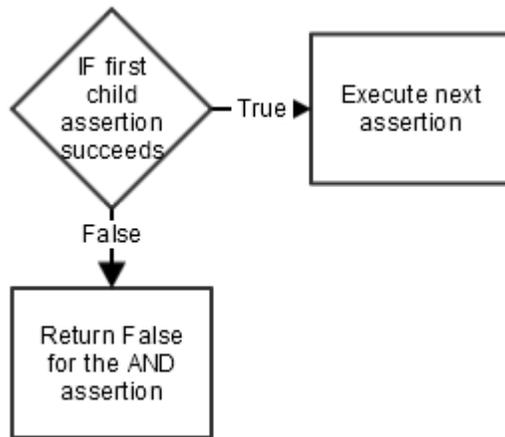
- [Run All Assertions Concurrently Assertion \(https://docops.ca.com/display/GATEWAY91/Run+All+Assertions+Concurrently+Assertion\)](https://docops.ca.com/display/GATEWAY91/Run+All+Assertions+Concurrently+Assertion): This is a variant of the [All Assertions Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion). All child assertions still must succeed for the parent to succeed, but the main difference is that all the children are run concurrently rather than sequentially.
- [Run Assertions for Each Item Assertion \(https://docops.ca.com/display/GATEWAY91/Run+Assertions+for+Each+Item+Assertion\)](https://docops.ca.com/display/GATEWAY91/Run+Assertions+for+Each+Item+Assertion): This is another tricky variant of the [All Assertions Must Evaluate to True Assertion \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion). The main difference here is that the child assertions may be run more than once or not at all, depending on the assertion's configuration.

Building Common Structures

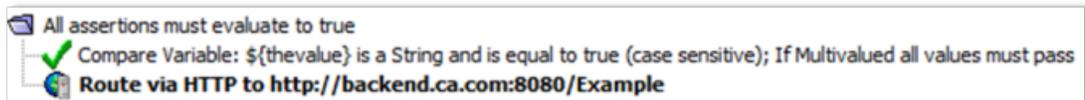
IF-THEN

The IF-THEN programming structure is easily reproduced using the AND assertion. The only possible confusion point for programmers is that strictly speaking, all inputs are evaluated in a logical AND but the Gateway stops processing the AND assertion at the first failure. Because of this, the Gateway's AND assertion can be thought of a "short-circuited" AND: the first child assertion that returns false causes the entire AND assertion to return false immediately.

To reproduce the IF-THEN in a service policy, use the AND assertion with two child assertions. If the first assertion succeeds, then the second assertion is run.



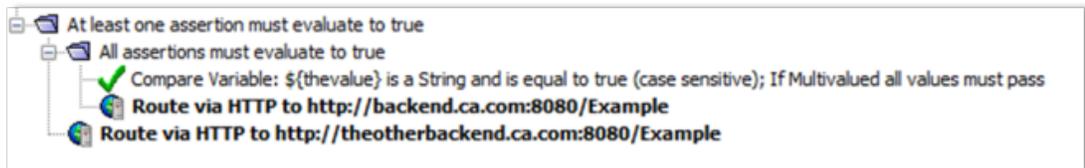
In the Policy Manager, this IF-THEN construct looks like the following. Note that composite assertions are represented by a folder icon, which implies the child assertions are evaluated in sequence.



You read this policy fragment as "If the context variable $\${thevalue}$ is equal to the string 'true', then attempt to route to the service located at <http://backend.ca.com:8080/Example>".

IF-THEN-ELSE

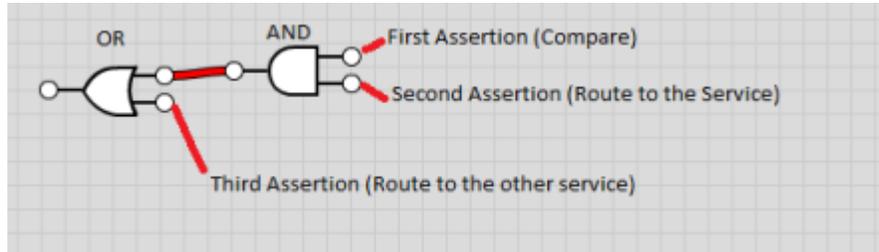
To replicate the IF-THEN-ELSE structure, embed the IF-THEN defined above within an OR assertion and then provide the alternate ELSE path:



Note the OR assertion ("At least...") is now the parent. The AND assertion ("All...") provides the IF-THEN logic. If this AND fails, the ELSE assertion (routing to "theotherbackend") is attempted. If the AND succeeds, the ELSE routing is not attempted because the OR assertion stops processing on the first successful child.

Reading the policy fragment: "If the context variable $\${thevalue}$ is equal to the string 'true', then attempt to route to the "backend" service, else attempt to route to the "otherbackend" service."

Using formal logic gates, this construct looks like this:

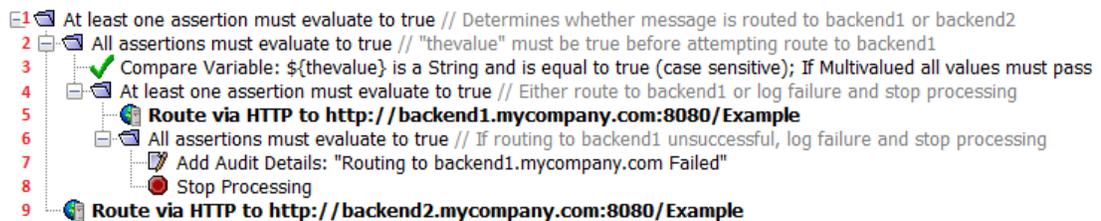


Composite Assertions Ramifications

Use composite assertions with care. They are powerful tools to help you precisely control your policy, but they can potentially hide errors. Here are some examples:

- The AND ("All...") assertion stops processing and returns false as soon as a child condition is not met. But it is not immediately clear what halted the processing: Is the comparison string not equal? Or an XPath did not find an element in the document? Or the back-end system did not acknowledge the HTTP request? Moreover, since processing is halted immediately, you will not know whether subsequent child assertions are successful.
- The OR ("At least...") assertion always runs the next assertion if the previous child returns false. You do not know if this false was caused by a logic check or whether the assertion returned an error. This means it is possible to have an error hidden by policy logic

Aside from hiding potential errors, it is easy to misread—or be misled by—nested composite assertions. Here is a simple example: You have primary ("backend1") and secondary ("backend2") routing destinations. If routing to the primary fails, you want to log an audit message and then attempt the secondary path. The proper way to achieve this is with a policy fragment like the following:



Reading the policy fragment: "If the context variable `${thevalue}` is equal to the string 'true', then attempt to route to the 'backend1' service. If that routing attempt fails, audit a message, and then stop and fail that sub-branch of the policy. Finally, if the previous policy sub-branch failed, attempt to route to 'backend2' service."

The Gateway always tracks the audit context for assertion errors, even if you don't include any auditing assertion in your policy. This means it is possible to have a service working as intended, hiding errors from users but reporting them to monitoring tools. This may be correct and desired outcome in most cases, but it can be unexpected and may cause some surprises.

The following table describes logic flow in the policy fragment, along with any ramifications.

Line	Description
1	This policy fragment is encased within an OR assertion. The two children are: the AND assertion in line 2 and the secondary routing in line 9.
2	The first child is an AND assertion. For this to succeed, both the comparison in line 3 <i>and</i> the OR assertion in line 4 must be true.
3	Tests the $\{thevalue\}$ variable. If it is true, routing to the primary back end (line 5) will be attempted, otherwise routing to the secondary (line 9) will occur.
4	An embedded OR assertion. Note that this will stop processing upon the first successful child (either line 5 or 6).
5	First child of the OR assertion in line 4. Routing is attempted to "backend1". If this is successful, processing of the entire policy fragment stops because the composite assertions in lines 4, 2, and 1 are satisfied.
6	Second child of the OR assertion in line 4. This is attempted only if routing in line 5 fails. Line 6 is another embedded AND assertion that is always designed to fail, because the assertion in line 8 will always fail.
<p> Tip: You may wonder, <i>Why do I need another composite assertion? Why can't I simply add line 7 right after line 5?</i> At first glance, this might be the easier solution: if the routing in line 5 fails, then simply log an audit saying so. However this causes another non-obvious logic error: replacing the composite assertion in line 6 with the assertion in line 7 will cause line 4 to always return 'true' (because the Add Audit Detail Assertion (https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion) always returns 'true'). As a result, this causes the composites in lines 2 and 1 to also return true, meaning the alternate routing in line 9 will never be attempted even when the primary in line 5 fails.</p>	
7	First child of the AND assertion in line 6: This adds a message to the audit log by using the Add Audit Detail Assertion (https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion) . This assertion always succeeds.
8	Second child of the AND assertion in line 6: The Stop Processing Assertion (https://docops.ca.com/display/GATEWAY91/Stop+Processing+Assertion) halts processing in that branch. This assertion always fails (that is returns 'false'). This is necessary to cause lines 6, 4, and 2 to return 'false', so that the alternate routing in line 9 is attempted.
9	Second child or the OR assertion in line 1; "backend2" is attempted only when the primary one fails.

Complex Policy Structures

The examples shown above are basic examples meant to provide a foundation to policy authoring. You can design policies with very complex interactions just by using the composite assertions, [encapsulated assertions \(https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions\)](https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions), and [policy fragments \(https://docops.ca.com/display/GATEWAY91/Policy+Fragments\)](https://docops.ca.com/display/GATEWAY91/Policy+Fragments). The key to success is to decompose problems into modular units and be careful with your policy logic.

CA API Gateway - 9.1

To illustrate the power of policy, several CA applications ([CA API Management OAuth Toolkit \(http://docops.ca.com/otk\)](http://docops.ca.com/otk) and the Mobile SSO capabilities within the [CA Mobile API Gateway \(http://docops.ca.com/mag\)](http://docops.ca.com/mag)) are delivered entirely as policy. Both of these products use policy fragments extensively to boost reliability and lower maintenance.

Policy Authoring Tips and Best Practices

Creating a policy that is easy to maintain and upgrade uses principles similar to good coding practices: make it readable and use comments generously. The following are some best practices that CA Technologies encourages in its training (may be superseded by local style guides):

- **Always include a comment block at the top of the policy**
This should describe the policy in general terms and define more specific terms, if required. To add a comment block, use the [Add Comment to Policy Assertion \(https://docops.ca.com/display/GATEWAY91/Add+Comment+to+Policy+Assertion\)](https://docops.ca.com/display/GATEWAY91/Add+Comment+to+Policy+Assertion). **Tip:** Comments display as one continuous line in the policy; they do not line wrap. For longer comments, consider using a series of "Add Comment" assertions.
- **Include versioning notes in the comment block**
Policies migrate through the enterprise and versioning notes are key to tracking changes. **Tip:** If you use revisions extensively, also consider adding a comment for each revision. For more information, see [Policy Revisions \(https://docops.ca.com/display/GATEWAY91/Policy+Revisions\)](https://docops.ca.com/display/GATEWAY91/Policy+Revisions).
- **Include information about passed arguments if policy is tied to encapsulated assertions, as well as any cluster-wide properties and global shared context variables .**
These should be in the comment block. The actual references are usually buried deep within the policy, so the policy is easier to comprehend if they are documented up front. **Tip:** Remember to update the comment as you add or remove references.
- **Use "right side" comments extensively to describe what is happening**
Many assertions hide their configurations inside their property dialogs, so the only way to understand what is happening is to open each assertion or (preferably) describe the configuration in a right-side comment. **Tips:** (1) The "right" and "left" comments are added through the [Add Comment to Policy Assertion \(https://docops.ca.com/display/GATEWAY91/Add+Comment+to+Policy+Assertion\)](https://docops.ca.com/display/GATEWAY91/Add+Comment+to+Policy+Assertion). (2) If your comments are not visible, click **Show Comments** in the [Policy Tool Bar \(https://docops.ca.com/display/GATEWAY91/Policy+Manager+Interface\)](https://docops.ca.com/display/GATEWAY91/Policy+Manager+Interface) or use the [View menu \(https://docops.ca.com/display/GATEWAY91/Policy+Manager+Menu\)](https://docops.ca.com/display/GATEWAY91/Policy+Manager+Menu).
- **Use "left side" comments sparingly**
The policy language is designed to read "VERB ACTION": "*Add comment...*", "*Authenticate against...*". Adding left side comments break up this readability. You should add left comments only to draw specific attention to a line, otherwise use right-side comments for everything.
- **Always start right-side comments with a consistent, familiar delimiter**
A common delimiter is '/', which is well known and improves the readability of the comment. **Tip:** Simply relying on the gray text applied to comments is often not clear enough. Moreover, policy fragments also display as gray text, causing it to be indistinguishable from comments added within a fragment.
- **Composite assertions should always have a comment**
Describe the intent of the logical AND ("[All assertions must... \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion)") and OR ("[At least one assertion... \(https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion\)](https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion)")

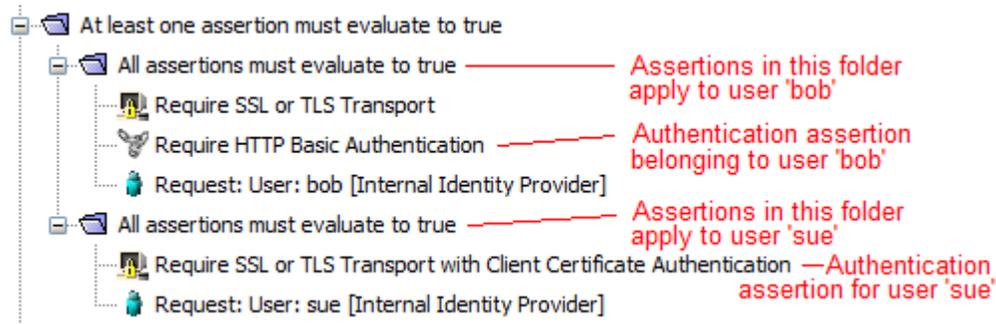
assertions in a comment. This is good practice, so that other users do not need to open the folder and analyze the contents to determine the intent of the composite assertion. **Tip:** For lengthier descriptions, use a series of [Add Comment to Policy Assertions](https://docops.ca.com/display/GATEWAY91/Add+Comment+to+Policy+Assertion) instead of one long right-side comment.

- **Use the "All assertions must..." folder to group functionality**
The collapsing nature of this folder (and the "At least..." folder) hides logical structures, while leaving comments visible. When collapsed, the policy may be more readable and it is easier to get the "big picture".
- **Prevent the "At least one..." folder from failing entire policy**
If you do not want the failure of an "At least one assertion..." folder to fail the entire policy, then add a [Continue Processing Assertion](https://docops.ca.com/display/GATEWAY91/Continue+Processing+Assertion) as a child assertion. This assertion always evaluates to true, preventing the policy from failing due a non-essential or conditional assertion failing.
- **Never use a template response to return an error message**
It may seem logical to create a custom response message using the [Return Template Response to Requestor Assertion](https://docops.ca.com/display/GATEWAY91/Return+Template+Response+to+Requestor+Assertion) when an error is encountered, but this is not the correct use of this assertion. Instead, always use the [Customize Error Response Assertion](https://docops.ca.com/display/GATEWAY91/Customize+Error+Response+Assertion) and fail the policy. This ensures that the Gateway is correctly notified of the error and records the failure correctly in the logs.
- **Turn on assertion line numbers**
Turn on assertion line numbers to help you edit or troubleshoot a policy. You can jump to a specific line number (Edit > Go to Assertion) or use the search feature (Edit > Find) to quickly locate an assertion. For more information, see [Assertion Numbering](https://docops.ca.com/display/GATEWAY91/Assertion+Numbering).
- **Ordering of assertions**
Some policy assertions work together and require the presence of each other to succeed. For example, the [Authenticate User or Group Assertion](https://docops.ca.com/display/GATEWAY91/Authenticate+User+or+Group+Assertion) requires an authentication assertion such as the [Require HTTP Basic Credential Assertion](https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion) to provide the credentials for validating the user's identity. Moreover, the authentication assertion must appear before the user or group. This is one example of how the presence and order of assertions can affect the ultimate validity of a policy.
- **Authentication assertion limitation**
An authentication assertion (such as the [Require HTTP Basic Credential Assertion](https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion)) can only provide credentials for a single user or group. The authentication assertion must appear before the identity provider assertion (for example, [Authenticate User or Group Assertion](https://docops.ca.com/display/GATEWAY91/Authenticate+User+or+Group+Assertion)).
- **Identity assertion recommendation**
It is best not to include more than one first-level identity assertion in a policy or within an "At least" or "All assertions" folder. If you must because the client expects the Gateway to authenticate more than one identity per request, the policy validator displays a warning but you

can still proceed.

To add more than one user or group in a policy, place each individual Authenticate User or Group assertion in a separate "At least one assertion" or "All assertions" folder with an authentication assertion (and other assertions, as required).

In the following figure, the credentials for user "Bob" are authenticated by the [Require HTTP Basic Credential Assertion](https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion) (<https://docops.ca.com/display/GATEWAY91/Require+HTTP+Basic+Credential+Assertion>), while "Sue" is authenticated by the [Require SSL or TLS Transport With Client Authentication Assertion](https://docops.ca.com/display/GATEWAY91/Require+SSL+or+TLS+Transport+With+Client+Authentication+Assertion) (<https://docops.ca.com/display/GATEWAY91/Require+SSL+or+TLS+Transport+With+Client+Authentication+Assertion>).



- It is best to put a [Stop Processing Assertion](https://docops.ca.com/display/GATEWAY91/Stop+Processing+Assertion) (<https://docops.ca.com/display/GATEWAY91/Stop+Processing+Assertion>) after any assertion whose sole purpose is to report on a prior error. Examples of such assertions include: [Audit Messages in Policy Assertion](https://docops.ca.com/display/GATEWAY91/Audit+Messages+in+Policy+Assertion) (<https://docops.ca.com/display/GATEWAY91/Audit+Messages+in+Policy+Assertion>), [Send Email Alert Assertion](https://docops.ca.com/display/GATEWAY91/Send+Email+Alert+Assertion) (<https://docops.ca.com/display/GATEWAY91/Send+Email+Alert+Assertion>), [Send SNMP Trap Assertion](https://docops.ca.com/display/GATEWAY91/Send+SNMP+Trap+Assertion) (<https://docops.ca.com/display/GATEWAY91/Send+SNMP+Trap+Assertion>), or [Return Template Response to Requestor Assertion](https://docops.ca.com/display/GATEWAY91/Return+Template+Response+to+Requestor+Assertion) (<https://docops.ca.com/display/GATEWAY91/Return+Template+Response+to+Requestor+Assertion>). Reason: These assertions always succeed, even though the intent is to halt the policy and send an HTTP challenge.
Example 1: You construct the following expecting the policy to halt if authentication fails, but in this case the routing will occur regardless.

```
Require HTTP Basic Credentials
At least one assertion must evaluate to true
  Request: Authenticate against Internal Identity Provider
  Audit: "Authentication Failed!"
Route
```

Example 2: To correct Example 1 so that the policy operates as intended, add a Stop Processing assertion:

```
Require HTTP Basic Credentials
At least one assertion must evaluate to true
  Request: Authenticate against Internal Identity Provider
  All assertions must be true
  Audit: "Authentication Failed!"
  Stop Processing
Route
```

- In general, assertions should be placed before the routing assertion in a policy. This is to ensure that all assertion conditions are met before the request is routed to the protected web service or XML application.
- Exceptions to the above are the assertions designed to operate on the response; these are examples of assertions that should be placed *after* the routing assertion:

- [Add Security Token Assertion \(https://docops.ca.com/display/GATEWAY91/Add+Security+Token+Assertion\)](https://docops.ca.com/display/GATEWAY91/Add+Security+Token+Assertion) (with target set to 'Response')
 - [Add Timestamp Assertion \(https://docops.ca.com/display/GATEWAY91/Add+Timestamp+Assertion\)](https://docops.ca.com/display/GATEWAY91/Add+Timestamp+Assertion) (with target set to 'Response')
 - [Apply XSL Transformation Assertion \(https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion\)](https://docops.ca.com/display/GATEWAY91/Apply+XSL+Transformation+Assertion)
 - [Encrypt Element Assertion \(https://docops.ca.com/display/GATEWAY91/Encrypt+Element+Assertion\)](https://docops.ca.com/display/GATEWAY91/Encrypt+Element+Assertion) (with target set to 'Response')
 - [Evaluate Response XPath Assertion \(https://docops.ca.com/display/GATEWAY91/Evaluate+Response+XPath+Assertion\)](https://docops.ca.com/display/GATEWAY91/Evaluate+Response+XPath+Assertion)
 - [Sign Element Assertion \(https://docops.ca.com/display/GATEWAY91/Sign+Element+Assertion\)](https://docops.ca.com/display/GATEWAY91/Sign+Element+Assertion) (with target set to 'Response')
 - [Validate XML Schema Assertion \(https://docops.ca.com/display/GATEWAY91/_Validate+XML+Schema+Assertion\)](https://docops.ca.com/display/GATEWAY91/_Validate+XML+Schema+Assertion)
- Assertions where you can specify the target message to be acted upon are prefixed with "Request:", "Response:", or "\${VARIABLE_NAME}" in the policy window. For example: *Request: Authenticate against XYZ or Response: Add signed Timestamp*. For more information, see [Select a Target Message \(https://docops.ca.com/display/GATEWAY91/Select+a+Target+Message\)](https://docops.ca.com/display/GATEWAY91/Select+a+Target+Message).
- Pay attention to Policy Validation Messages window. It displays helpful messages as you configure or validate a policy.
- Use the Copy and Paste options on the Edit menu to help you organize the policy.
- Instead of [deleting an assertion \(https://docops.ca.com/display/GATEWAY91/Add%2C+Delete%2C+Enable%2C+Disable+Assertions\)](https://docops.ca.com/display/GATEWAY91/Add%2C+Delete%2C+Enable%2C+Disable+Assertions), consider [disabling \(https://docops.ca.com/display/GATEWAY91/Add%2C+Delete%2C+Enable%2C+Disable+Assertions\)](https://docops.ca.com/display/GATEWAY91/Add%2C+Delete%2C+Enable%2C+Disable+Assertions) it instead. Disabling an assertion is useful during testing and troubleshooting. It has the same effect as deleting the assertion, but you can easily restore the assertion by re-enabling it,
- If you disable all assertions in a ["All assertions... \(https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion\)"](https://docops.ca.com/display/GATEWAY91/All+Assertions+Must+Evaluate+to+True+Assertion) folder, this folder will succeed. However if you disable all assertions within a ["At least one... \(https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion\)"](https://docops.ca.com/display/GATEWAY91/At+Least+One+Assertion+Must+Evaluate+to+True+Assertion) folder, this folder will fail

The following is an example policy that uses many of the best practices described above:

CA API Gateway - 9.1

test [/test] (v15/15, active)

Save and Activate Save Validate Export Policy Import Policy Import From UDDI Hide Comments Hide Assertion Numbers

```
2 Comment: *****
3 Comment: * Policy for "OTK Access Token Retrieval" encapsulated assertion
4 Comment: *
5 Comment: * Find the access_token within the http header, http query parameter
6 Comment: *
7 Comment: * Passed Argument: ${allow_header} // boolean: Allow Authorization Header
8 Comment: * Passed Argument: ${allow_query} // boolean: Allow parameter
9 Comment: * Passed Argument: ${auth_header} // string: Authentication header
10 Comment: * Passed Argument: ${auth_param_token} // string: Parameter
11 Comment: * Passed Argument: ${given_access_token} // string: Access token (bearer only)
12 Comment: * Returned Argument: ${access_token}
13 Comment: * Returned Argument: ${auth_header}
14 Comment: * Returned Argument: ${auth_scheme}
15 Comment: *
16 Comment: * MAG Version: 2.2.01
17 Comment: * Modified by JayMac to add general failure response and MAC token validation - 20150120
18 Comment: *****
19 Comment: This policy can be found and should be used as 'Encapsulated Assertion!'
20 Comment: == Allow the auth-header OR parameters but NOT both if both are used
21 Add Audit Details: "=> Starting ${policy.name} fragment"
22 ✓ Set Context Variable auth_scheme as String to empty
23 ✓ Set Context Variable hasError as String to: false
24 At least one assertion must evaluate to true // Extract token and assign to ${access_token} for return to policy
25 All assertions must evaluate to true // Locate as Parameter
26 ✓ Compare Variable: ${allow_query} is equal to true (case sensitive); If Multivalued all values must pass
27 ✓ Compare Variable: ${auth_header} is empty (case sensitive); If Multivalued all values must pass
28 ✓ Compare Variable: ${auth_param_token} is a String and is not empty (case sensitive); If Multivalued fail assertion
29 ✓ Set Context Variable auth_scheme as String to: Bearer
30 ✓ Set Context Variable access_token as String to: ${auth_param_token}
31 All assertions must evaluate to true // Locate in HTTP header
32 ✓ Compare Variable: ${allow_header} is equal to true (case sensitive); If Multivalued all values must pass
33 ✓ Compare Variable: ${auth_param_token} is empty (case sensitive); If Multivalued all values must pass
34 ✗ ${auth_header}: Evaluate Regular Expression - ^b([Bb][Ee][Aa][Rr][Ee][Rr])MAC\b\s(.{1,256})$
35 ✓ Set Context Variable auth_scheme as String to: ${schema_token[1]}
36 ✓ Set Context Variable access_token as String to: ${schema_token[2]}
37 At least one assertion must evaluate to true // Get MAC access_token if MAC profile is used
38 All assertions must evaluate to true // Process for MAC profile
39 ✓ Compare Variable: ${auth_scheme} is equal to MAC; If Multivalued all values must pass
40 ✗ ${access_token}: Evaluate Regular Expression - id\s*"=*\s*([^\s]*)
41 ✓ Compare Variable: ${id[1]} is not empty; If Multivalued all values must pass
42 ✓ Set Context Variable access_token as String to: ${id[1]}
43 ✓ JayMac Compare Variable: ${auth_scheme} is not equal to MAC; If Multivalued all values must pass // Continue only if it wasn't a MAC profile
44 All assertions must evaluate to true // Use passed parameter in ${given_access_token}
45 ✓ Compare Variable: ${given_access_token} is not empty; If Multivalued all values must pass
46 ✓ Set Context Variable access_token as String to: ${given_access_token}
47 ✓ Set Context Variable auth_scheme as String to: Bearer
48 At least one assertion must evaluate to true // Fail - Missing or invalid token
49 All assertions must evaluate to true // Could not find token in header, parameter or passed
60 All assertions must evaluate to true // Multiple access token
74 JayMac All assertions must evaluate to true // Invalid access_token (general fail)
82 Add Audit Details: "=> Leaving ${policy.name} fragment"
```

Policy Performance Optimization

How a policy is constructed plays a large role in performance. A prime example is a policy during development vs. the same one deployed into production. You may implement verbose logging and extensive auditing during development to help you troubleshoot. But in production, such logging can obscure the real error and make it more difficult to determine the root cause of issues. Also, extensive auditing will impact Gateway performance and consume disk space rapidly.

- [Logging and Auditing \(see page 31\)](#)
 - [Auditing is Very Expensive \(see page 31\)](#)
 - [Logging is Cheaper, But Not Free \(see page 32\)](#)
- [Back-End Latency \(see page 32\)](#)
 - [Latency vs. TPS vs. Concurrency \(see page 32\)](#)
 - [Network Performance \(see page 33\)](#)
- [CPU-Intensive Operations \(see page 33\)](#)
 - [Per-Request SSL Session Initiation \(see page 33\)](#)
 - [Cryptographic Assertions \(see page 33\)](#)
 - [Create or Sign SAML Assertion \(see page 33\)](#)
 - [Issue JSON Web Token \(see page 34\)](#)
- [Data Transformation \(see page 34\)](#)
 - [Mode Switching \(see page 34\)](#)
 - [XSL Transforms \(see page 34\)](#)
 - [Regular Expressions \(see page 35\)](#)
 - [JSON Transform \(see page 35\)](#)
- [Caching \(see page 35\)](#)
- [Bottlenecks in Common Policy Elements \(see page 36\)](#)
 - [Database Operations \(see page 36\)](#)
 - [LDAP Operations \(see page 36\)](#)
 - [CA SSO and other SSO products \(see page 36\)](#)

Performance optimization means accomplishing a task in the most efficient way. Often, this means focusing on reducing latency:

- Latency slows down the individual API calls, providing slower individual responses to client software.
- Policies with high latency limits throughput, as this causes message processing threads to remain open until the thread completes. Sometimes more concurrency can alleviate this, but other times additional concurrency only increases CPU load with more task switching overhead.

Latency has two costs:

- *Wait time*: The Gateway idles while waiting for external resources to complete their work.

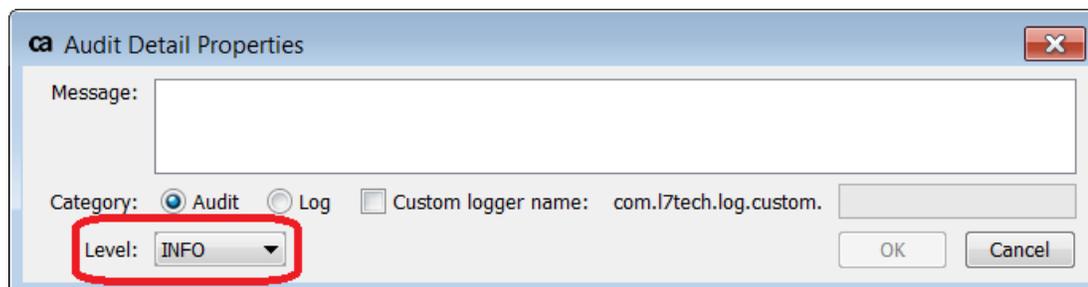
- *Local CPU usage:* If a policy's latency is caused by an overtaxed CPU, then this limits the number of requests that the Gateway can process over time. External latency is often unavoidable but local CPU usage is often in the control of the policy author .

Logging and Auditing

Auditing is Very Expensive

Use auditing sparingly, as it can impact Gateway performance, sometimes significantly. During the development/testing phase, detailed audits can help you identify and resolve problem points. However when the policy is moved to production, remove all unnecessary audits to maximize Gateway performance. CA Technologies strongly recommends against using the audit subsystem unless mandated by regulatory agencies.

Training material often recommend using the [Add Audit Detail Assertion](https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion) (<https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion>) to help develop your policy. The default action of this assertion is to audit at the INFO level:



This is a useful tool for policy development, but is a detriment to production as the INFO level will generate many audits. You should delete all instances of the [Add Audit Detail Assertion](https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion) (<https://docops.ca.com/display/GATEWAY91/Add+Audit+Detail+Assertion>) and [Audit Messages in Policy Assertion](https://docops.ca.com/display/GATEWAY91/Audit+Messages+in+Policy+Assertion) (<https://docops.ca.com/display/GATEWAY91/Audit+Messages+in+Policy+Assertion>) for production policies, except for noting errors and debug situations. They should never be used as part of normal policy flow. Ideally, a production policy should produce no audits at all during successful execution.

A more technical look at auditing:

Auditing requires additional writes to disk. In high concurrency production environments this introduces "external latency", as the Gateway waits for the audit subsystem to complete. This in turn prevents the working thread from being returned to the thread pool. A typical audit may take between 50 to 500 milliseconds to complete and each audit creates a multi-table database insert transaction to the database. Thus, the maximum insert rate of the database used for auditing effectively becomes the maximum auditing rate, which in turn is limited by the disk I/O performance. For standard hard drives, this is measure in hundreds per second; solid-state drives offer better performance but it is still measure in the thousands per second. In clustered environments, write rates become even more crucial as it limits cluster scaling to the cluster-wide database.

If auditing is unavoidable, audit to external databases as they may offer better performance and scaling compare to the on-device database. The Logging subsystem is preferred for audit-like actions.

Logging is Cheaper, But Not Free

In the Add Audit Details Properties shown above, you can opt to **Log** transactions rather than audit them. This causes the Gateway to capture the data in a log file rather than initiating a database transaction. This is less "expensive" than auditing as it requires less overhead.

Logs lacks the structure and viewing user experience of our [audit viewer \(https://docops.ca.com/display/GATEWAY91/View+Gateway+Audit+Events\)](https://docops.ca.com/display/GATEWAY91/View+Gateway+Audit+Events), but they do not have the database transaction limitations. The Gateway still flushes to the hard disk at intervals, so the IOPS rating of the disk is still important but is much less limiting than audits.

It is important to understand the consequences of logging: If you log 30 messages per policy at a desired 1000 transactions per second, then you are attempting to write 30,000 messages to disk per second. If these are short messages, there are two effects:

1. The disk space used.
2. The "noise level" of these informational messages.

There is no need to trace the operation of every successful request. During production debugging, all the successful requests make it difficult to find the useful error messages. You could refer to this as "Signal to noise ratio".

Large and frequent logging messages might also reach disk write bandwidth limitations. This is much less common but has occurred in several field deployments.

As with auditing, CA Technologies recommends that a production policy logs nothing at all in the successful case. This reduces the signal to noise ratio and does not waste disk space or time. If you need to log success cases, keep the logging data to a single line per successful request.

Back-End Latency

Back-end systems play a large part of the overall performance of an API and back-end latency is often the cause of perceived Gateway performance issues .

Latency vs. TPS vs. Concurrency

Transactions per second (TPS) is defined as the number of whole policy executions that can complete in one second. It is the inverse of latency. For example, if an API call (including network and Gateway time) to the backend takes 100 milliseconds to complete, then a single requestor could execute 10 of those calls in one second. Examining the metrics data, you discover:

- The policy execution took 10 milliseconds
- The network required another 10 milliseconds
- Waiting for the back-end response took the remaining 80 milliseconds

If your expectation is that the API must be able to provide 1000 transactions per second, then 100 such requests must be performed in parallel to equal 1000 TPS. In reality, the back end may not sustain 80ms responses times—it could be longer. This increased back-end latency drives the need for even more concurrency to sustain the 1000 TPS expectations. Large back-end latency creates the perception that the API is slow and that the Gateway is the bottleneck. It is important to consider latency during any performance evaluation.

Network Performance

Network (Wide Area, Metropolitan Area) performance can also contribute to overall latency. Bandwidth could be a factor during certain time of day, especially when large message are involved. Or it could just be distance: round trip between Los Angeles and New York City even at the speed of light requires about 30ms.

CPU-Intensive Operations

As mentioned earlier, CPU usage can cause latency. You can avoid this with careful choices around policy composition, understanding of the relative costs, and careful balancing of business requirements against the reality of scaling.

Per-Request SSL Session Initiation

Negotiating SSL sessions is very expensive. For this reason, we recommend [HTTP keep alive](https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion#RouteviaHTTP(S)Assertion-keepalive) ([https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion#RouteviaHTTP\(S\)Assertion-keepalive](https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion#RouteviaHTTP(S)Assertion-keepalive)) and [SSL Session reuse](https://docops.ca.com/display/GATEWAY91/Configuring+the+Load+Balancer#ConfiguringtheLoadBalancer-sslreuse) (<https://docops.ca.com/display/GATEWAY91/Configuring+the+Load+Balancer#ConfiguringtheLoadBalancer-sslreuse>) via either SSL Session affinity at the load balancer or other techniques.

Cryptographic Assertions

In general, cryptography is an expensive operation. Some use cases absolutely require it, but you can avoid the most expensive operations in most others.

For example, validating a signature requires less overhead than creating one and signature validation is good security practice. In some token exchange use cases, you can avoid creating token signatures by caching outbound tokens on a per-identity basis. This reduces the overhead of signing or encrypting data.

Create or Sign SAML Assertion

SAML signatures are an asymmetric cryptographic operation that demands high CPU processing requirements. As a result, they can incur significant latency and slow down policy execution. It is not uncommon to see delays of up to hundreds of milliseconds, depending on the algorithms involved. Use outbound token caching for improved performance.

Issue JSON Web Token

JSON Web Tokens are the direct equivalents of SAML in the JSON world, with exactly the same issues surrounding CPU consumption.

Data Transformation

Mode Switching

The Gateway specifically treats XML data using DOM parsing and reduces our XML overhead by preserving the DOM data structure for the duration of policy execution. For policies that perform XML manipulation using XSL or Schema Validation, or XPath, the Gateway attempts to avoid re-parsing data by operating on the preserved DOM data structure. This methodology helps maintain performance.

By comparison, you force the Gateway into needless mode switching by doing something like the following:

1. You first manipulate the `${request}` message (or other message type variables) with XML operators.
2. Then you manipulate it with string operators such as the [Evaluate Regular Expression Assertion](https://docops.ca.com/display/GATEWAY91/Evaluate+Regular+Expression+Assertion) (<https://docops.ca.com/display/GATEWAY91/Evaluate+Regular+Expression+Assertion>) and [Set Context Variable Assertion](https://docops.ca.com/display/GATEWAY91/Set+Context+Variable+Assertion) (<https://docops.ca.com/display/GATEWAY91/Set+Context+Variable+Assertion>).
3. Later in the policy, you operate again on the same message data with XSL, XML Schema, or XPath manipulation or inspection.
4. All this causes the Gateway to re-parse the message data because regular expressions and other string operators cannot operate on the DOM structure, which forces it to modify the string representation.
5. As a result, the Gateway must re-render the XML DOM data as a String, then re-parse it back into a DOM.

XSL Transforms

XSL transformations are less of a concern than cryptography, but they can still cost 9 ms per request for a 10 Kbyte message. This scales up with larger messages and puts additional pressure on the Gateway's "garbage collection" subsystem.

Regular Expressions

Though lighter weight in terms of memory consumption than XSL, regular expressions still can have CPU-related performance issues.

Some relatively common regex patterns are surprisingly more "expensive": In general, avoid back references (of the form '?=' and other similar patterns) as that causes much larger CPU usage since several standard optimizations are disabled by that construct.

JSON Transform

This is currently not quantified, but has the same heavy string data manipulation as regular expressions.

Caching

Caching can help in many situations where there is a bottleneck on a crucial resource.

You can leverage in-policy caching to help many external dependencies. Be aware that inbound pure round-robin caching can defeat certain types of caching, so you need to plan inbound load balancing carefully.

LDAP concurrency can occur in certain policies; the [Query LDAP Assertion \(https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion\)](https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion) is often used for non-standard LDAP validation.

The built-in [LDAP Identity Provider \(https://docops.ca.com/display/GATEWAY91/LDAP+Identity+Providers\)](https://docops.ca.com/display/GATEWAY91/LDAP+Identity+Providers) has specific cache sizing configuration. Even though the [Query LDAP Assertion \(https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion\)](https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion) does not use the same cache, you can configure a number of LDAP cache settings from within the assertion's properties.

Caching can even help avoid CPU bottlenecks in certain instances. For example, consider a use case involving token transformation. Without caching, the policy issues a new JSON Web Token (JWT) with every request, even though the inbound user credential token did not change. The CPU overhead of the JWT was significant enough to cause a slowdown. Caching the fully created and signed JWT increases performance dramatically. This is because the overhead of creating and maintaining cache entries is still lower than the cryptographic signature cost.

In general, look for expensive operations that happen on a per-request basis without providing new information as prime candidates for caching.

Bottlenecks in Common Policy Elements

This section describes common bottlenecks that have single points of dependency. Any policy that has crucial external single points of dependency on every request will be constrained by the performance of that single point. A common approach to avoid this is to use caching and mitigation strategies.

Database Operations

Using the [Perform JDBC Query Assertion \(https://docops.ca.com/display/GATEWAY91/Perform+JDBC+Query+Assertion\)](https://docops.ca.com/display/GATEWAY91/Perform+JDBC+Query+Assertion) has transaction rate limitations similar to auditing, because of the transactional nature of the connection. As each policy cannot have knowledge of other concurrent operations, the Gateway cannot easily do multi-statement transactions. This means you should avoid database operations.

On a related note, the API Management OAuth Toolkit has a corollary feature: its default configuration uses a database for token storage. These tokens have longer lifetimes, resulting in less tokens being issued over time, leading to lower token insert rates.

LDAP Operations

The Gateway's [LDAP Identity Provider \(https://docops.ca.com/display/GATEWAY91/LDAP+Identity+Providers\)](https://docops.ca.com/display/GATEWAY91/LDAP+Identity+Providers) is heavily cached to improve performance. You can adjust settings using the [LDAP Cluster Properties \(https://docops.ca.com/display/GATEWAY91/LDAP+Cluster+Properties\)](https://docops.ca.com/display/GATEWAY91/LDAP+Cluster+Properties) for larger cache sizes and connection pools to support higher transaction rates and larger concurrency. The [Query LDAP Assertion \(https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion\)](https://docops.ca.com/display/GATEWAY91/Query+LDAP+Assertion) has its own cache size and age, so that needs to be checked against transaction rates and concurrency.

CA SSO and other SSO products

As of Gateway v9.0, the CA Single Sign-On (formerly "CA SiteMinder") capability does not have connection pooling and decision caching. If you require these capabilities, contact CA to obtain the Tactical version of this assertion. The default behavior of CA Single Sign-On is synchronous and heavily dependent on the performance of the CA SSO policy server, which does not scale at the same rate as a cluster of Gateways.

Turning Use Cases into Policies and Fragments

This topic shows how you can convert common use cases into service policies and policy fragments on the CA API Gateway.

- [Creating a Service and Service Policy \(see page 37\)](#)
 - [The Simple Case: Authentication and HTTP Routing Assertion \(see page 38\)](#)
 - [Gathering Credentials \(see page 38\)](#)
 - [Tokens \(see page 38\)](#)
 - [Validating Credentials/Authentication \(AuthN\) \(see page 38\)](#)
 - [Authorization \(AuthZ\) \(see page 39\)](#)
 - [LDAP \(see page 39\)](#)
 - [CA Single Sign-On and Other SSO Products \(see page 39\)](#)
 - [Routing \(see page 39\)](#)
- [Mediation Cases \(see page 39\)](#)
 - [Transport Mediation \(see page 40\)](#)
 - [Token Type Mediation \(see page 40\)](#)
- [Federation \(see page 40\)](#)
- [Policy Fragments and Encapsulated Assertions \(see page 41\)](#)
- [Auditing \(see page 41\)](#)

Creating a Service and Service Policy

The CA API Gateway can publish a variety of different services. These can be broadly categorized as:

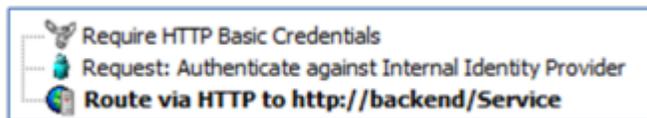
- **SOAP services:** A common beginning task is to locate the WSDL file for a service and then "publish" it. If a WSDL is not available, a wizard is available to help you create one.
- **Non-SOAP services:** REST and JSON are examples of non-SOAP services that are rapidly gaining in popularity. One possible disadvantage with these services is that less metadata are available. This requires more effort from the policy architect to pull in the various metadata. As a result, JSON excels at publishing simple services, while requiring considerably more effort to create a complex service with strong validation.

The Simple Case: Authentication and HTTP Routing Assertion

The simple use case is also very common. A policy for this just involves three policy elements: Gather credentials, validate the credentials, and finally send the original request to a back end server. In this case, we're allowing all valid users to access the back end resource, so Authorization is implied by Authentication. The final act of the Gateway returning the back end server's response to the user is implied by the policy finishing successfully.

As there's no optional events and no decisions that result in anything other than "This is not a valid request", there's no need for any composite assertions, no branches into conditional states. The policy looks like this:

Our simple example policy requires no composite assertions nor conditional branching. There are no optional events nor decisions resulting in anything other than "This is not a valid request". The sample looks like this in the Policy Manager:



Gathering Credentials

The CA API Gateway supports many types of credentials, including: usernames and passwords, various single sign-on tokens, X.509 certificates, SAML tokens, JWT, and other more obscure ones.

Credentials must be available in the policy before an assertion that validates them executes. That underscores the sequential nature of policy execution: the Gateway cannot validate credentials if it has not extracted them from the inbound context yet.

Tokens

Validating Credentials/Authentication (AuthN)

Validation implies mostly the authentication step, but some tooling, like RADIUS implies both Authentication and Authorization. It is best practice to validate all credentials. The Policy Manager warns you when you extract credentials without validating them.

There may be instances where gathering credentials and validating them with custom tooling triggers the warning—for example, the policy validates credentials via direct database lookup. If this happens, ignore the warning. You can still save the policy even when validation warnings are present.

Note that gathering and validating credentials is done via data structures that are always available, often referred to as the "message context". This is largely automatic, but there are cases where specifically referring to this context is done via context variables. For more information about these variables, see [Authentication Context Variables \(https://docops.ca.com/display/GATEWAY91/Authentication+Context+Variables\)](https://docops.ca.com/display/GATEWAY91/Authentication+Context+Variables).

Authorization (AuthZ)

Authorization can be thought of as either explicit or implicit. *Explicit* authorization examines the results of authentication for certain attributes. Implicit authorization uses an external decision tool such as CA Single Sign-on, or a local authorization tool built by the organization.

LDAP

LDAP is very common in organizations. These are organized into "LDAP Identity Providers", which provide a convenient way to authenticate and extract attributes. These attributes are then used in decision trees in the policy.

CA Single Sign-On and Other SSO Products

Note that SSO tools often combine both Authorization and Authentication into a single call. This results in cleaner, simpler-looking policies, but at the expense of "dividing" the policy: in the Gateway as a policy enforcement point and in the SSO tool as a policy decision point. This separation may cause some difficulties, but there are good reasons based on corporate governance to keep the SSO tool-based decision point.

Routing

Sending a validated request to a backend service is the most common deployment of the CA API Gateway. The Policy Manager provides a variety of routing assertions: see [Message Routing Assertions \(https://docops.ca.com/display/GATEWAY91/Message+Routing+Assertions\)](https://docops.ca.com/display/GATEWAY91/Message+Routing+Assertions) for details. In the most common scenarios, the Gateway authorizes access to the back end, to pass request credentials, role accounts, or mutually-authenticated SSL.

Mediation Cases

The CA API Gateway is designed as a message processing engine that is optimized for HTTP use cases, but is not exclusively a "HTTP gateway". The message processor accepts any message that enters the flow and simply executes the policy until it exits. Policy authors use the AND/OR logic rules to construct a policy with specific input/output needs and the Gateway processes all the steps.

Transport Mediation

A relatively common use case involves receiving a message over HTTP and then sending the message to a queue like MQ Series or Tibco EMS. The policy for this is nearly identical to the common case shown previously, but instead of the [Route via HTTP\(S\) Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+HTTP%28S%29+Assertion), you use the [Route via JMS Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+JMS+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+JMS+Assertion) or [Route via MQ Native Assertion \(https://docops.ca.com/display/GATEWAY91/Route+via+MQ+Native+Assertion\)](https://docops.ca.com/display/GATEWAY91/Route+via+MQ+Native+Assertion).

Token Type Mediation

Expanding on the simple case, you can add a policy step to create or obtain a different kind of credential or token once credentials are validated. This can be as simple as HTTP Basic credentials inbound, or SOAP with WS-Basic Authentication to the back end. The Policy Manager provides assertions for token creation and single sign-on authorization.

You can add the credential tokens to any of the following:

- the response message
- the response HTTP context (in the case of browser-style interactions)
- the request message or context (in the case of SOAP or REST style interactions)

Note that the Gateway does not require redirects to add HTTP cookies to the backend request context, because the Gateway does this work via direct requests and specific policy configuration. It generally does not involve the browser.

Federation

Federation in the Gateway can be simply token mediation: Instead of a simple token, a cryptographic token is generated in policy, like a SAML token or JSON web token.

The Gateway can also request from an external token service. Example: The Gateway submits a WS-Trust message to a SAML STS (Secure Token Service). It receives the response, parses the SAML token, and then places the token into the WS-Security header of the SOAP message. The Gateway then sends the combined message to the real back end service.

The flow is conceptually the same in JSON: replace the WS Trust with whatever the JWT STS needs as a request,. The JSON web token and structure of the message is manipulated as JSON.

The Gateway Federation is strictly claims based, without requiring a backing identity. The decision to issue a token comes from the policy. It can be derived from identity or simply from the credential presented.

Policy Fragments and Encapsulated Assertions

Policy fragments are standard policy logic designed to work with more than one service. A fragment also referred to as an "included policy" or simply "include". Programmers can liken them to library routines, with strictly defined inputs and outputs and limited side effects.

Policy fragments can have [role-based access control](https://docops.ca.com/display/GATEWAY91/Manage+Roles) rules, similar to service policies. This allows them to be authored by domain experts, while being read-only to the policy authors. This provides the same benefits as standard software development: domain experts define common policy elements (for example, like Authentication, Authorization, Audit) and policy authors simply insert the appropriate fragments into their policies. Fragments help reduce errors and simplifies testing, validation, and governance.

A close variation of policy fragments are [encapsulated assertions](https://docops.ca.com/display/GATEWAY91/Encapsulated+Assertions). These can be viewed as "home made" assertions that is powered by an underlying policy fragment. The main difference is that the policy fragment is completely concealed within an encapsulated assertion, accepting input from the user interface crafted for that assertion.

Encapsulated assertions allow you to strictly define your inputs and outputs, and are a critical tool to create strong, locally-relevant best practices.

Auditing

Deciding on auditing is a critical part of developing any policy. Business concerns often require complete auditing, but be aware that auditing is not "free". There are serious implications to consider when auditing every request. The default audit setup consumes significant disk space, which may be in limited supply on appliance Gateways (physical or virtual). Carefully consider your auditing early in the service deployment plan.

Keep in mind these core concepts for auditing:

- Auditing both request and response content is disk space intensive, making it unsuitable for high volume services. For these, consider a simple audit without the request and response body.
- The Gateway normally returns the response before flushing the audit. As the thread does not return to the pool until the audit flushes, the performance is equivalent to "fully synchronous". This means the audit subsystem could be a bottleneck in high concurrency use cases and performance tests.
- Adding queuing can make auditing more asynchronous, but certain queues are also synchronous to disk by default.

Implementing a Policy Development Life Cycle

A traditional Software Development Life Cycle (SDLC) focuses on configuration management, essential for tracking and predicting which artifacts are included in a production build. A Policy Development Life Cycle (PDLC) employs similar concepts and tooling. You need to keep tracking of versions and perform rollbacks if necessary. There are phases for development, testing, and deployment. However the PDLC is further complicated by differences in environments; for example, you cannot reliably test a Gateway policy for a user acceptance environment by connecting to a development LDAP server. You need to connect to an LDAP server that is configured for production.

Developers working in the SDLC will find much familiarity in the PDLC. Just treat your configuration artifacts (policies and their dependencies) as source code, specifically XML. The PDLC has a specific phase that deals with resolving environmental dependencies.

- [Terminology \(see page 42\)](#)
- [Policies are Code and Deserve Similar Attention \(see page 43\)](#)
- [Best Practices \(see page 43\)](#)
- [GUI-Based Policy Migration Using the Policy Manager \(see page 44\)](#)
- [Automated Policy Migration Using Management APIs, GMU, CMT \(see page 44\)](#)
 - [CMT/GMU \(see page 44\)](#)
- [Conclusion \(see page 45\)](#)

Terminology

- *Service* refers to the API the Gateway presents to the users. This could be standard SOAP services, REST services, JSON, or various kinds of networked XML and non-XML APIs.
- *Environmental Dependencies* are the specific components that a service depends upon, for example: Public Keys, Private Keys, LDAP configurations, JMS configurations, JDBC configurations, MQ Series configurations, etc. The dependency will be referenced by name when discussing details.
- *DEV* refers to the initial policy development environment.
- *QA* refers to the initial quality assurance environment.
- *UAT* refers to the acceptance testing and/or performance testing environment.
- *PROD* refers to the deployed environment in front of production services and APIs.

Policies are Code and Deserve Similar Attention

In enterprise deployments that include infrastructure components such as the CA API Gateway, it is critical to manage configuration artifacts in a repeatable and reliable way. Most organizations are familiar with deploying software to application servers. This section shows you how to leverage this knowledge to deploy policies to infrastructure.

CA Technologies provides tools and interfaces for an organization to plan a life cycle around for their environment. As every organization has its own way of deploying production environments, some parts of this process will already be in place. What is important is the effect of these processes, not the exact steps.

Best Practices

CA Technologies has compiled the following valuable tips, based on many deployments:

- Develop a policy and its environmental dependencies in a test (i.e., non-production) environment.
- Copy the complete version of these artifacts to a test environment with no shared infrastructure
- Make the local changes needed to be compatible with the test environment.
- Local changes should be environment specific (for example, IP addresses, host names of Application Servers or LDAP servers).
- Test the system to be sure your components are working.
- Copy the same components to the next step in the deployment pipeline.
- The original developer should not be responsible for moving artifacts to production, nor be responsible for testing.
- Part of good development practice is to have repeatable creation of your systems.
- If documentation isn't sufficient to do full deployment, fix the documentation. Do not assign the original developer.
- Certain kinds of environment specific artifacts, like host names of back end servers, should use cluster properties instead of literals.
- Naming conventions for objects can ease resolution of environment dependencies.
- For predictability, configuration must be pushed as a unit, and the tested versions of each artifact be used. Version control features like tags and/or branches can help.
- Local dependencies are worked out by moving from development to testing or user acceptance.
- Moving directly from DEV to PROD is risky, as testing the dependency discovery is critical.

- Never make changes in PROD. Always do changes in DEV and move to QA, etc.
- Do not share environment items like LDAP between DEV and UAT.
- Sharing environments without a clear goal in mind will almost certainly cause a failure moving to production.

GUI-Based Policy Migration Using the Policy Manager

The Policy Manager provides a simple way of migrating policies between environments via its `import` (<https://docops.ca.com/display/GATEWAY91/Import+a+Global+Resource>) and `export` (<https://docops.ca.com/display/GATEWAY91/Exporting+a+Policy>) capabilities. However, these methods do not preserve mapping between multiple environments, making them less suitable for larger deployments and multiple policy development groups.

Automated Policy Migration Using Management APIs, GMU, CMT

For environments that do not allow GUI tooling for managing production systems, the Gateway provides several command line options:

- APIs for remote management (`WS-Man` (<https://docops.ca.com/display/GATEWAY91/WS+Management+API>) and `RESTman` (<https://docops.ca.com/display/GATEWAY91/REST+Management+API>)) and their accompanying command line clients
- The `Gateway Migration Utility` (<https://docops.ca.com/display/GATEWAY91/Gateway+Migration>) (GMU)
- The older `Command line Policy Migration Toolkit` (<https://docops.ca.com/display/GATEWAY91/Gateway+Migration>) (CMT) (see how CMT maps to GMU [here](https://docops.ca.com/display/GATEWAY91/CMT+Differences+and+Command+Mappings) (<https://docops.ca.com/display/GATEWAY91/CMT+Differences+and+Command+Mappings>)).

For more information, see `Develop Migration-Friendly Policies`. (<https://docops.ca.com/display/GATEWAY90/Develop+Migration-Friendly+Policies>)

CMT/GMU

Gateway versions 8.3 and later include the Gateway Migration Utility (GMU) to cover policy migration use cases. Earlier versions of the Gateway are supported by Command line Policy Migration Toolkit (CMT), which is also available in the later versions.

For GMU and CMT, version control is critical. As locally preferred tooling (Subversion, ClearCase, Polytron Version Control, Git, etc.) is widely varied, CA Technologies cannot prescribe a specific tool, but rather expects three main capabilities of a version control system:

- Can check in and check out based on a "tag" or "release name" that encompasses many files and many directories
- Can deal with XML as text
- Can be run using command line tooling

Internally, the Gateway uses Object Identifiers (OIDs) and Global Object Identifiers (GOIDs) to identify various configuration items. When dealing with multiple environments and multiple staff members being involved in the process, name-based identifiers are beneficial. The GMU tooling deals with resolving names to local objects.

Similarly, develop naming conventions to identify the back-end App Servers and any artifacts.

Conclusion

Creating a Policy Development Life Cycle that mirrors the software development life cycle at your organization is the first step to a reliable process. The Gateway provides the tools and capabilities that can adapt to any environment. Planning is essential. Once the process is in place, it can and should be fully automated.

The Gateway's policy language supports diverse use cases. Plan for environment dependencies, choose good modularity, and above all think in policy in the same way as source code.