Gateway Performance Troubleshooting

Summary

Troubleshooting the CA API Gateway requires understanding of how the system is put together, what diagnostic tools and capabilities exist, and which to deploy at what times.

Audience

This guide assumes basic familiarity with the installation, configuration, and use of the Gateway. For further information about these topics, please see the Gateway installation guide and the policy authoring guide.

Purpose

This guide explains how the Gateway works and what facilities are available for diagnosing problems that occur.

Gateway Theory of Operation

Overview

The CA API Gateway is a multi-protocol message-based gateway that accepts inbound messages (such as HTTP requests) via a variety of inbound transports and applies policies to them. Commonly, a policy will enforce checks on a request and then cause the Gateway to forward the message on to another endpoint (possibly via a different transport) and forward any response back to the original requestor.

An individual Gateway node runs as multiple threads within a single process hosting a Java virtual machine. There is a separate host process called the Process Controller that starts the Gateway node process and monitors it, managing its software configuration and restarting the Gateway if it crashes. Policies and other configuration relevant to message processing is stored in a database, which can either be embedded in the Gateway node process or else an external process. Multiple Gateway nodes sharing the same external database comprise a Gateway cluster. Text menu-based tools are provided for initial low level configuration. Once a Gateway node is running, it can be configured and managed using the Policy Manager thick client or web applet.

Configuration Tools

Configuration menus are provided to configure the Gateway including creating or upgrading the database and, for the appliance or virtual appliance, to configure networking.

Database

The Gateway stores almost all of its configuration information in a database. The database can be a MySQL database running in an external process (possibly on a remote host) or it can be an embedded Derby database.

Information Not Included in the Database

Not all configuration information can be stored in the database. Some is stored on disk on the node instead.   Most obviously, information about what kind of database to use and how to connect to it is stored in the node.properties file.

Of particular relevance to troubleshooting are the cluster passphrase and the master passphrase.

The cluster passphrase, stored as the node.cluster.pass property in the node.properties file (in the Gateway/node/default/etc/conf directory), is used to decrypt the cluster shared key from the database.

The master passphrase is used to encrypt passwords and secrets (those with values starting with "$L7C$") stored in properties files on disk. If the Gateway is configured to use an nCipher HSM, the master passphrase is stored the kmp.properties file, encrypted for a private key in the HSM. Otherwise, it is stored in obfuscated form in the omp.dat file.

Information Stored in the Database

The information in the Gateway database breaks down into the following categories:

Read-mostly configuration

This accounts for most of the tables in the Gateway database and contains information such as:

- cluster properties
- transport configuration and firewall rules
- services and policies
- folders and aliases
- encapsulated assertions
- security zones and assignments
- trusted certificates and revocation checking policies
- RBAC permissions and role assignments
- UDDI registries and services
- Cluster shared key
- configuration objects used by add-on modules

Mutable configuration

Some configuration objects may be changed autonomously by the Gateway when certain events occur. This can include:

- Email POP3 last poll time and message ID
- UDDI proxied service current publish status
- UDDI registry subscriptions
- UDDI business service status and runtime information

Key Store

The Gateway may be configured to store private key material in an external hardware security module such as Thales nCipher or SafeNet HSM. If not so configured, it will store them in the database protected with the cluster shared key.

External MySQL Database

If a database connection is configured, the Gateway will connect to a MySQL database (either on the same or a different machine). This is currently the only way to have a cluster of Gateway nodes sharing the same configuration database.

Embedded Derby Database

If no external database connection is configured the Gateway will use an internal Derby database and will run with service metrics disabled.

Process Controller

The Process Controller is a Java process that launches and monitors the main Gateway Java process.

Gateway

The Gateway is a Java process that hosts the actual policy engine (message processor) as well as the inbound and outbound transports and other features controllable via policy and configured by the Gateway's configuration database.

Processing Model

Each request is processed in its own thread. A transport thread receives a request and invokes the message processor, which typically does not return until the request has been completely handled (including back end routing).

Request Flow

Inbound Transport

The lifecycle of a request begins when an inbound transport (such as an HTTP or FTP listen port, or a JMS or email listener) gets a request message.

The inbound transport thread creates what is internally called a “policy enforcement context” and invokes the message processor component.

Global policies

If a message-received global policy exists, it is run first, before any service resolution or other policy processing, and before the request has been examined (for transports such as HTTP, this happens even before the request body has been received). Custom service resolution can optionally be performed by this global policy.

Service Resolution

The message processor examines the request to see which service policy should be applied. This may involve just checking the request URL, or (for SOAP services sharing a URL) may require deeper examination of the request body. The Gateway may need to decrypt an encrypted SOAP body in order to perform resolution.

Service Policy

After resolution completes, the corresponding compiled service policy is evaluated. (If any pre-security or pre-service global policies exists and have not yet been executed then they will be evaluated first, at this point.)

A service policy is a collection of assertions which are evaluated in order. Some assertions may be composite assertions, such as All or One Or More, which evaluate child assertions of their own.

The service policy succeeds if all top-level assertions in the policy succeed.

Routing Assertions

Routing a request to a back-end protected service and collecting the response is done by a routing assertion such as the HTTP Routing assertion.

Auditing

At the end of request processing a message summary audit record is created. Depending on the Gateway’s audit configuration the record may be saved to the internal audit store or may be given to the audit sink policy for further processing.

Gateway Troubleshooting

Gateway Ping Endpoint

Gateway listen ports may enable a Ping service (listed under Built-in services in the Listen Port Properties dialog).

When enabled, the Ping service is available on the URI /ssg/ping

The Ping service, by default, is available only over HTTPS, and requires authentication. It shows the Gateway node’s uptime and status.

A secondary URI can provide additional information about the operating system and virtual machine, including current node memory usage.

Extra Debug Info (Stack Traces)

Generic Java (Gateway, Manager)

Jps

The jps utility is a command line tool that comes with the Java runtime environment. It lists Java processes running on the same machine.

You should use the version of jps from the JDK version (and bit width, i.e. 32 bit vs 64 bit) that is running the target Java process.

The utility must be run as the same OS-level user ID as the target process (or be run as root).

Jstack

You can use the jstack utility to obtain a thread dump from a running Java process. See the JVM Level section below for more information about jstack.

Gateway

The Gateway will log additional stack traces (that are normally suppressed as unneeded) if run with the Gateway system property `com.l7tech.logging.debug` set to “true”.

This may work when added to the Gateway’s system.properties file, or (if this doesn’t take effect early enough) may require adding the property directly to the Gateway’s Java command line such as by dropping a new file into the Gateway’s profile.d directory.

Manager

The Policy Manager also supports logging additional stack traces to its own log if the Manager system property `com.l7tech.logging.debug` is set to “true”.

This may require editing the Manager.ini file.

Manager Applet

To enable stack traces for failed admin API calls in the applet, open the Java control panel on the PC running the browser. Go to the Java tab -> View... button -> User tab and edit the "Runtime Parameters" for the JDK version used to run the applet. Add the argument `"-Dcom.l7tech.logging.debug=true"`. Click Ok. In the Advanced tab, ensure "Java console" is set to "Show console". Confirm the dialogs and restart your browser.

As the applet launches, click in the java console and press the "5" key to increase the log level.

You can confirm that stack traces are now being logged for failed remote invocations by trying to delete an active audit sink policy; it should result in a stack trace in the log after "WARNING: Exception during remote API call: com.l7tech.policy.PolicyDeletionForbiddenException".

Then, if you can reproduce the policy manager disconnect error again, there should be a stack trace in the log showing what happened.

Make sure you add the `"-Dcom.l7tech.logging.debug=true"` option to the JDK version actually being used by the browser. The version number shown in the row in the parameters table should match the JDK version shown at the beginning of the Java debug console. If the system-wide Java control panel is controlling the wrong JDK version, you can go directly to the control panel for the version of Java you want by navigating to the JDK directory's "bin" subdirectory and then launching the "javacpl.exe" utility from there.

Gateway Log Files

The Gateway produces log files during normal operation that can be used to monitor its operation as well as to diagnose problems that occur. Additional log files can be configured using the Manage Log Sinks option in the Policy Manager GUI.

Process Controller Log Files

The Process Controller has its own log file that records its activity. This is particularly useful for checking to see whether the Process Controller has killed and restarted a Gateway process for being non responsive to its pings.

Configuration Log Files

The configuration menus may record logs during process such as database upgrades that can be useful to diagnose issues should they occur.

Support Info Script

A feature is provided to gather information useful to the CA API Gateway support team, such as versions of installed components.

Audit Messages in Policy Assertion

The Audit Messages in Policy assertion can be used to boost the audit level of a policy, possibly only when certain conditions occur. Boosting the audit level can cause an audit record to be recorded in the audit sink when the request has finished processing.

Add Audit Details Assertion

The Add Audit Details assertion can be added to a policy to record additional information to the log or to the audit sink. This can include context variables.

Policy Tracing

The Policy Tracing feature can be enabled on a service policy to record extremely detailed information about the activity of the policy.

When Policy Tracing is enabled for a service, a special Trace policy is executed after every policy assertion in the traced policy. The trace policy can thus record the assertion path that is taken along with policy state and context variables as they are changed during execution.

Policy Step Debugger

The Policy Manager can be used to step through execution of a service policy as it handles a request. You can place breakpoints or single-step through the policy, observing the policy state and context variables as they change.

Watchdog Assertion

A Watchdog assertion is available from CA Support that can be used to log a stack trace for policy execution threads that fail to complete processing of a request in a reasonable amount of time.

OS Level

The underlying operating system that runs the Gateway provides numerous monitoring and diagnostic features that can shed light on the behavior of the Gateway (along with the rest of the system).

top

The top command line utility shows which processes are using CPU time and reveals what the CPU is doing.

vmstat

The vmstat command line utility shows what the OS’s virtual memory subsystem is doing. This is particularly useful to see if the OS has run out of memory and begun to page information to disk. This should normally never occur on a properly configured Gateway host.

lsof

The lsof command line utility shows files, pipes and sockets that are currently in use by processes such as the Gateway.

JVM Level

Stack Trace

A stack trace is often included with an error message or in a log file. The stack trace shows exactly what a thread was doing at the time an error occurred.

The stack trace shows an execution path that led to the current position in the code.

If the stack trace is due to an error or exception, more than one position might be shown. The first position is that of the code that most recently handled the error (often by logging it or generating the error message containing the stack trace). Then there may be a line starting with "Caused by:" followed by the position of an older the exception or error that caused the newer one. This may be repeated until the last cause in the list, which is (hopefully!) the root cause of the error.

For each position, the path through the code is read from bottom to top.

Gateway Example – Simple Stack Trace

Here is an example of a stack trace logged by a Gateway with extra debug logging enabled when an attempt is made to save an HTTP Routing assertion whose policy XML has been manually modified to include an invalid URL:

```
WARNING: Invalid protected service URL: unknown protocol: httpf
java.net.MalformedURLException: unknown protocol: httpf
         at java.net.URL.<init>(URL.java:592)
         at java.net.URL.<init>(URL.java:482)
         at java.net.URL.<init>(URL.java:431)
         at
com.l7tech.server.policy.assertion.ServerHttpRoutingAssertion.<init>(ServerHttpRouting
Assertion.java:114)
         at sun.reflect.GeneratedConstructorAccessor215.newInstance(Unknown Source)
         at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccesso
rImpl.java:45)
         at java.lang.reflect.Constructor.newInstance(Constructor.java:526)
         at
com.l7tech.server.policy.ServerPolicyFactory.doMakeServerAssertion(ServerPolicyFactory
.java:189)
         at
com.l7tech.server.policy.ServerPolicyFactory.compileSubtree(ServerPolicyFactory.java:1
22)
         at
com.l7tech.server.policy.assertion.composite.ServerCompositeAssertion.<init>(ServerCom
positeAssertion.java:60)
         at
com.l7tech.server.policy.assertion.composite.ServerAllAssertion.<init>(ServerAllAssert
ion.java:27)
         at sun.reflect.GeneratedConstructorAccessor210.newInstance(Unknown Source)
         at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccesso
rImpl.java:45)
         at java.lang.reflect.Constructor.newInstance(Constructor.java:526)
         at
com.l7tech.server.policy.ServerPolicyFactory.doMakeServerAssertion(ServerPolicyFactory
.java:189)
```

```
         at
com.l7tech.server.policy.ServerPolicyFactory.access$000(ServerPolicyFactory.java:29)
         at
com.l7tech.server.policy.ServerPolicyFactory$2.call(ServerPolicyFactory.java:96)
         at
com.l7tech.server.policy.ServerPolicyFactory$2.call(ServerPolicyFactory.java:93)
         at
com.l7tech.server.policy.ServerPolicyFactory.doWithEnforcement(ServerPolicyFactory.jav
a:49)
         at
com.l7tech.server.policy.ServerPolicyFactory.compilePolicy(ServerPolicyFactory.java:10
1)
         at
com.l7tech.server.policy.PolicyCacheImpl.buildServerPolicy(PolicyCacheImpl.java:705)
         at
com.l7tech.server.policy.PolicyCacheImpl.findDependentPolicies(PolicyCacheImpl.java:11
49)
         at
com.l7tech.server.policy.PolicyCacheImpl.updateInternal(PolicyCacheImpl.java:933)
         at
com.l7tech.server.policy.PolicyCacheImpl.notifyUpdate(PolicyCacheImpl.java:758)
         at
com.l7tech.server.policy.PolicyCacheImpl.onApplicationEvent(PolicyCacheImpl.java:557)
         at
com.l7tech.server.util.PostStartupApplicationListener$StartupListenerRegistration$1.on
ApplicationEvent(PostStartupApplicationListener.java:62)
         at
org.springframework.context.event.SimpleApplicationEventMulticaster.multicastEvent(Sim
pleApplicationEventMulticaster.java:97)
         at
org.springframework.context.support.AbstractApplicationContext.publishEvent(AbstractAp
plicationContext.java:303)
         at
com.l7tech.server.EntityVersionChecker.dispatchInvalidation(EntityVersionChecker.java:
225)
         at
com.l7tech.server.EntityVersionChecker.access$900(EntityVersionChecker.java:30)
         at
com.l7tech.server.EntityVersionChecker$DispatchingTransactionSynchronization$1.run(Ent
ityVersionChecker.java:369)
         at
com.l7tech.server.util.ManagedTimer$ManagedTimerTaskWrapper.doRun(ManagedTimer.java:24
1)
```

```
          at com.l7tech.server.util.ManagedTimerTask.run(ManagedTimerTask.java:30)
          at java.util.TimerThread.mainLoop(Timer.java:555)
          at java.util.TimerThread.run(Timer.java:505)
```

This example has no chained causes and can simply be read from bottom to top. It shows:

- The thread began execution as a TimerThread. This likely means it belongs to a background task of some kind.
- It next invokes the EntityVersionChecker. This is a component that periodically checks to see whether entities have been changed in the Gateway's configuration database.
- The EntityVersionChecker invokes the PolicyCacheImpl component via a published event. (publishEvent, then onApplicationevent). This indicates that an updated entity was detected in the database. Since the PolicyCacheImpl is responding to the event, the updated entity is most likely a Policy.
- The PolicyCacheImpl is invoking the ServerPolicyFactory component to compile a Policy.
- A ServerHttpRoutingAssertion constructor is being invoked. This is the result of compiling an HTTP Routing assertion within a Policy.
- The routing assertion constructor is attempting to parse a URL using the Java URL class.
- This parse is failing with a MalformedURLException with the exception message "unknown protocol: httpf"

From this we determine that the problem is that a policy contains a misconfigured HTTP Routing assertion whose routing URL is invalid because it starts with "httpf" instead of either "http" or "https".

Thread Dump

A thread dump is essentially a stack trace for every thread currently running within a Java process. This sheds light on what the program is doing.   If a thread is waiting for something to happen before it can be proceed, this will often reveal what it is. Similarly, if OS level utilities have shown the Java process to be spinning its wheels consuming CPU, a thread dump can reveal what it is trying to do.

For each thread, the process of identifying what it was doing at the time the dump was taken is essentially the same as the process of reading a stack trace. Refer to How to Capture and Analyze Thread and Heap Dumps for more information.

jstack

You can use the jstack utility to obtain a thread dump from a running Java process. Refer to How to Capture and Analyze Thread and Heap Dumps for more information.

Heap Histogram

A heap object histogram shows a count of object types in a Java process's heap memory. This is much smaller than a full heap dump but can be useful for investigating memory exhaustion or memory leaks.

Jmap

The jmap utility is a command line tool that comes with the Java runtime environment. Refer to How to Capture and Analyze Thread and Heap Dumps for more information.

Heap Dump

A heap dump records the entire contents of a Java process's heap memory. This includes all Java objects that have been allocated and not yet garbage collected. Refer to How to Capture and Analyze Thread and Heap Dumps for more information.

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath="/tmp/heap.hprof"

A Java process can be started with special options to cause a heap dump to be recorded automatically should an OutOfMemoryError occur.

This has all the save caveats as recording a heap dump using the jmap utility.

jvisualvm

The jvisualvm utility is a GUI application that provides information about Java processes running on the same machine. You should use the version of jvisualvm from the JDK version that is running the target Java process. The utility must be run as the same OS-level user ID as the target process (or be run as root).

Use on a Headless Server

Because a GUI is required, running jvisualvm on a headless Linux server requires that X client libraries be installed, and that an X server (screen/keyboard/mouse) be available to host the GUI. If you are using ssh to connect to the host running the target Java process from a host running an X server (such as a Linux or Solaris GUI client machine), you can use SSH port forwarding to allow jvisualvm to connect back through to your X server through the SSH connection. The jvisualvm utility, discussed above, can connect to a running Java process and show a variety of useful diagnostic information.

JMX operations

The Java management extensions can be used by a utility such as jvisualvm to do fine-grained monitoring and control of certain objects within a Java process. The Gateway exposes some of its internal objects via JMX.

Rebuild Policy Cache

Using JMX it is possible to force the Gateway to rebuild its policy cache without having to restart.

Profiling

A profiler such as jvisualvm can be used to determine how a running Java process is spending its CPU and memory budget.

prof

The argument –prof can be added to a java command line to cause it to record CPU usage statistics while the Java program runs. At the end of execution, a .prof file will be saved in the current directory.

Types of Issues

A variety of issues may affect a Gateway.

Network Level

The network is a common source of trouble.

IP No Route to Host

This error indicates that connectivity to the destination network has been lost.

TCP Connection Timed Out

This error could mean that connectivity has been lost or that the target host is down.

This could also mean that a connection is blocked by a firewall.

TCP Connection Refused

This error indicates that the target host is reachable but that the server process is not running or the connectivity has been blocked by a firewall.

Gateway Inbound Transport

Errors that occur while trying to push messages into the Gateway will manifest in the Gateway’s inbound transport.

HTTP(S) Connection Timed Out

If the TCP connection succeeds, but there is a timeout at the HTTP(S) level when connecting to a Gateway, the HTTP transport could be out of threads. A thread dump will show if all the HTTP pool threads are busy.

This could also happen if the Gateway is being starved for CPU. In this case the top utility will show what is consuming CPU.

TLS Server Hello Not Received

If the TCP connection succeeds, but the client times out before a TLS server hello message is sent from the server, the cause could be similar to one of the causes of an HTTP(S) connection time out.

Gateway Message Processor

Some errors may occur after a request is received but before server policy evaluation begins.

Service Not Found

If the Gateway is unable to identify a policy that applies to the request, the request will be rejected. This can happen if more than one service could apply to a request. For example, if multiple SOAP services on the same URL exist using the same WSDL. The Policy Manager will try to prevent you from publishing a second conflicting service in this case.

Service Policy Not Valid

If a request is resolved to a service that does not currently have a valid policy then the request will be rejected. A policy is invalid if it (or an include it relies on) could not be compiled due to a configuration problem. The Policy Manager will try to prevent you from saving policies in an invalid state.

Gateway Policy Logic

Sometimes a request arrives and is dispatched to a valid policy, but the policy itself contains logic errors.

Tools to use:

Logging via audit details

You can place Add Audit Details assertions into a policy at key locations to record the value of context variables. You may need an Audit Messages in Policy assertion to boost the audit level so that the message summary audit record containing these added audit details actually gets saved.

Policy Tracing

A trace policy can be used to do examine the state of a policy context after every assertion has completed. You can enable policy tracing using a checkbox in the Service Properties dialog for a published service. The default trace policy adds an audit detail after every assertion that completes showing the assertion and its completion status.

Policy Step Debugger

The policy step debugger can be used to step through policy execution one assertion at a time as it processes a request. You can set breakpoints and view context variables while the policy is paused. To debug a service, right-click the service node and select “Service Debugger” from the context menu.

See the policy authoring manual for more information about this feature.

Gateway Outbound Routing

Errors very frequently occur while the Gateway is trying to route an outbound request to a protected service.

Performance

Sometimes everything works, but the system is slower than required.

Out Of Memory

The Gateway can run out of memory. This should be avoided. Tools are provided to limit the Gateway’s memory use.

The first line of defense is concurrency and message size limits: the minimum memory cost just to process messages concurrently is the concurrency limit times the message size limit (given certain assumptions). The concurrency is controlled by the inbound transports. The message size limit is controlled by cluster properties. The Gateway can also run out of memory if features such as the cache assertion are used without tuning to limit the size of the cache.