US009270621B1

(12) **United States Patent**
Muehlebach et al.

(10) **Patent No.: US 9,270,621 B1**
(45) **Date of Patent: Feb. 23, 2016**

(54) **SECURELY PROVIDING MESSAGES FROM THE CLOUD**

(71) Applicant: **CA, Inc.**, Islandia, NY (US)

(72) Inventors: **Heidi R. Muehlebach**, Melbourne (AU); **Trent Fitzgibbon**, Melbourne (AU); **Brad Gossit**, Melbourne (AU)

(73) Assignee: **CA, Inc.**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 350 days.

(21) Appl. No.: **13/776,395**

(22) Filed: **Feb. 25, 2013**

(51) **Int. Cl.**
G06F 15/16 (2006.01)
H04L 12/58 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 51/00*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,359,978	B2	4/2008	Hinde et al.	
8,181,238	B2	5/2012	Holar et al.	
8,745,710	B1 *	6/2014	Roth	H04L 63/08 726/6
8,903,884	B2 *	12/2014	Vasters	G06F 9/5011 709/201
2005/0076126	A1	4/2005	Knight et al.	
2006/0236387	A1 *	10/2006	Ballinger	H04L 63/08 726/14
2010/0125899	A1	5/2010	Tinnakornsrisuphap et al.	
2010/0287278	A1	11/2010	Edwards	
2011/0264765	A1 *	10/2011	Devadhar	G06F 9/546 709/217
2012/0042216	A1 *	2/2012	Blubaugh	H04L 63/029 714/48
2012/0144024	A1 *	6/2012	Lee	G06F 21/41 709/224
2012/0265976	A1	10/2012	Spiers et al.	
2012/0281706	A1	11/2012	Agarwal et al.	
2013/0191500	A1 *	7/2013	Shafi	H04L 29/0809 709/217
2013/0286951	A1 *	10/2013	Viswanathan	H04W 4/18 370/329
2013/0326346	A1 *	12/2013	Zhu	H04L 67/10 715/260
2013/0326487	A1 *	12/2013	Yousouf	G06F 8/30 717/134
2014/0075446	A1 *	3/2014	Wang	G06F 9/50 718/104
2014/0101110	A1 *	4/2014	Rittle	G06F 17/30165 707/654
2014/0156684	A1 *	6/2014	Zaslaysky	G06F 17/30389 707/756
2014/0157113	A1 *	6/2014	Krishna	G06F 17/289 715/249
2014/0337474	A1 *	11/2014	Khuti	H04L 41/5038 709/217

FOREIGN PATENT DOCUMENTS

WO	2005/089063	A2	9/2005

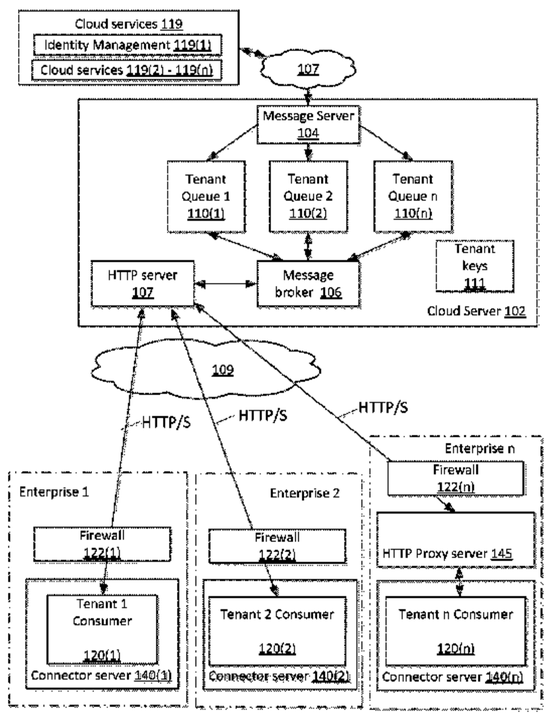
* cited by examiner

Primary Examiner — Mohamed Wasel
Assistant Examiner — Phyllis A Book
(74) *Attorney, Agent, or Firm* — Vierra Magen Marcus LLP

(57) **ABSTRACT**

Securely providing messages from a cloud server to an enterprise is disclosed. In one aspect, a message is received at a first server that provides a cloud service to a plurality of tenants. The message complies with a protocol other than HTTP. The message is put on a message queue for a first tenant of the plurality of tenants. An HTTP connection is established in response to a request from a second server to establish the HTTP connection between the second server and the first server. The first server receives an HTTP request from the second serve over the HTTP connection for a message for the first tenant. The message is provided to the second server over the HTTP connection in response to determining that the second server is authorized to receive messages for the first tenant.

21 Claims, 8 Drawing Sheets



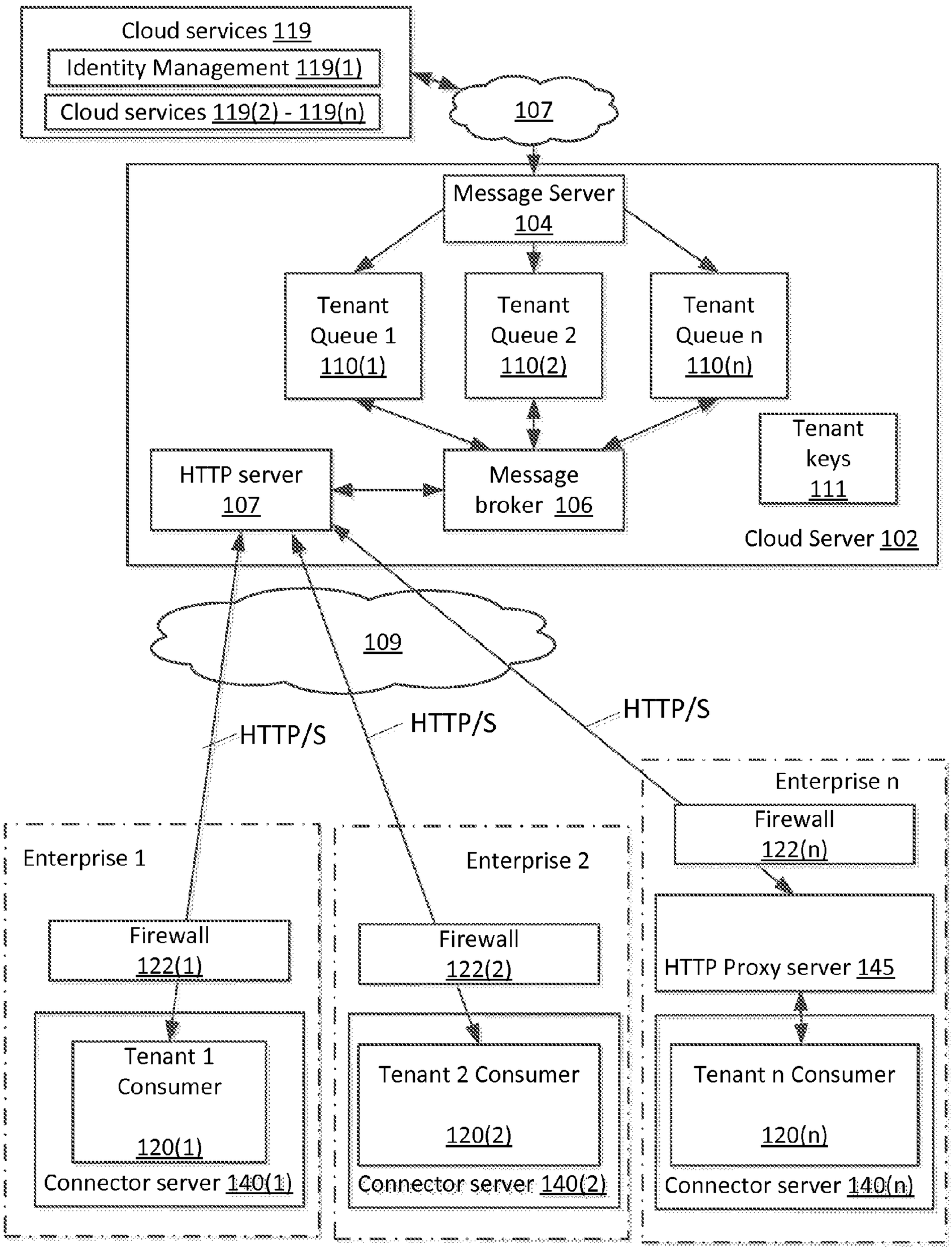

Cloud services 119
Identity Management 119(1)
Cloud services 119(2) - 119(n)
107
Message Server 104
Tenant Queue 1 110(1)
Tenant Queue 2 110(2)
Tenant Queue n 110(n)
HTTP server 107
Message broker 106
Tenant keys 111
Cloud Server 102
109
HTTP/S
HTTP/S
HTTP/S
Enterprise 1
Firewall 122(1)
Tenant 1 Consumer 120(1)
Connector server 140(1)
Enterprise 2
Firewall 122(2)
Tenant 2 Consumer 120(2)
Connector server 140(2)
Enterprise n
Firewall 122(n)
HTTP Proxy server 145
Tenant n Consumer 120(n)
Connector server 140(n)

Figure 1

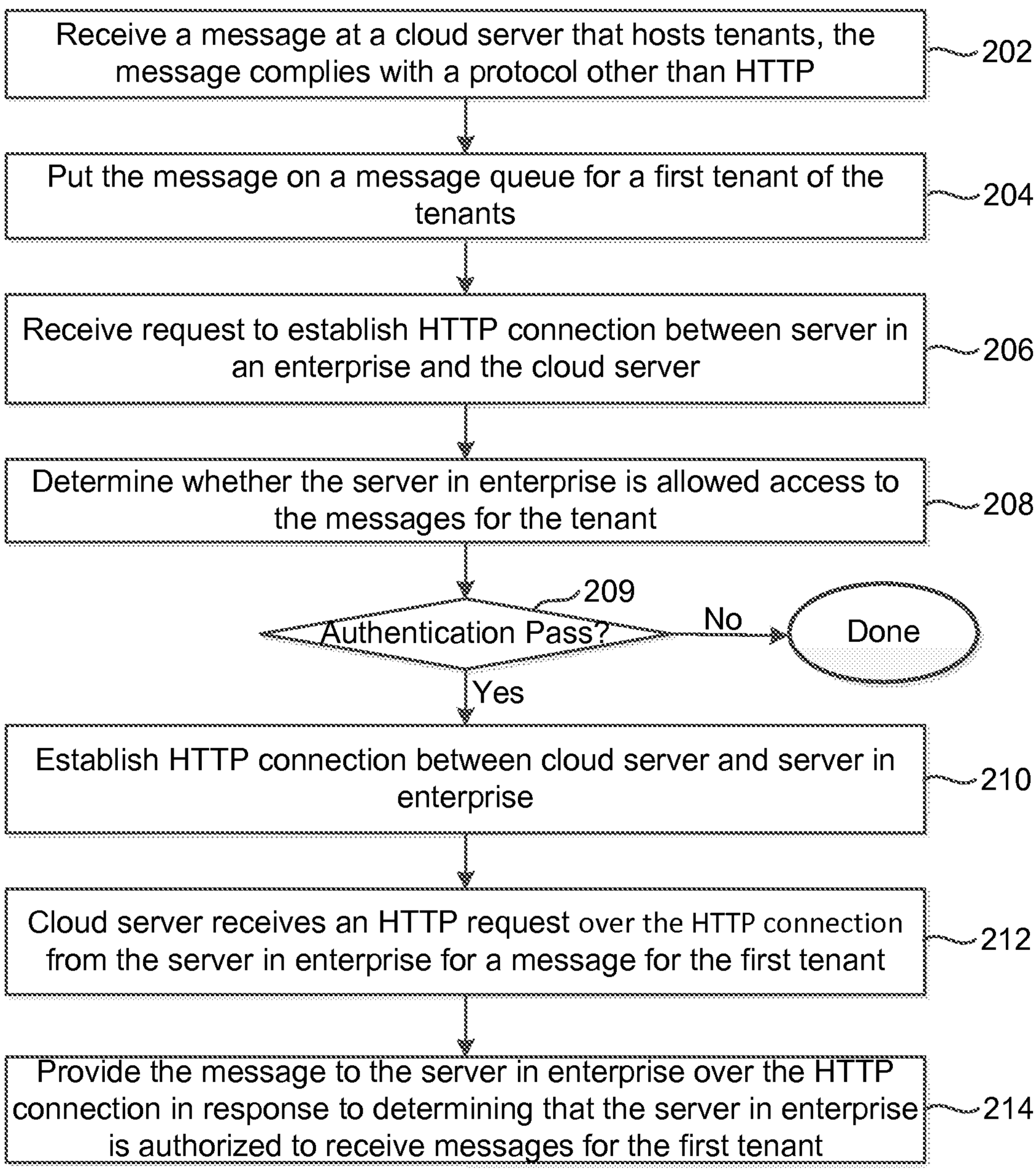

200
Receive a message at a cloud server that hosts tenants, the message complies with a protocol other than HTTP
202
Put the message on a message queue for a first tenant of the tenants
204
Receive request to establish HTTP connection between server in an enterprise and the cloud server
206
Determine whether the server in enterprise is allowed access to the messages for the tenant
208
209
Authentication Pass?
No
Done
Yes
Establish HTTP connection between cloud server and server in enterprise
210
Cloud server receives an HTTP request over the HTTP connection from the server in enterprise for a message for the first tenant
212
Provide the message to the server in enterprise over the HTTP connection in response to determining that the server in enterprise is authorized to receive messages for the first tenant
214

Figure 2

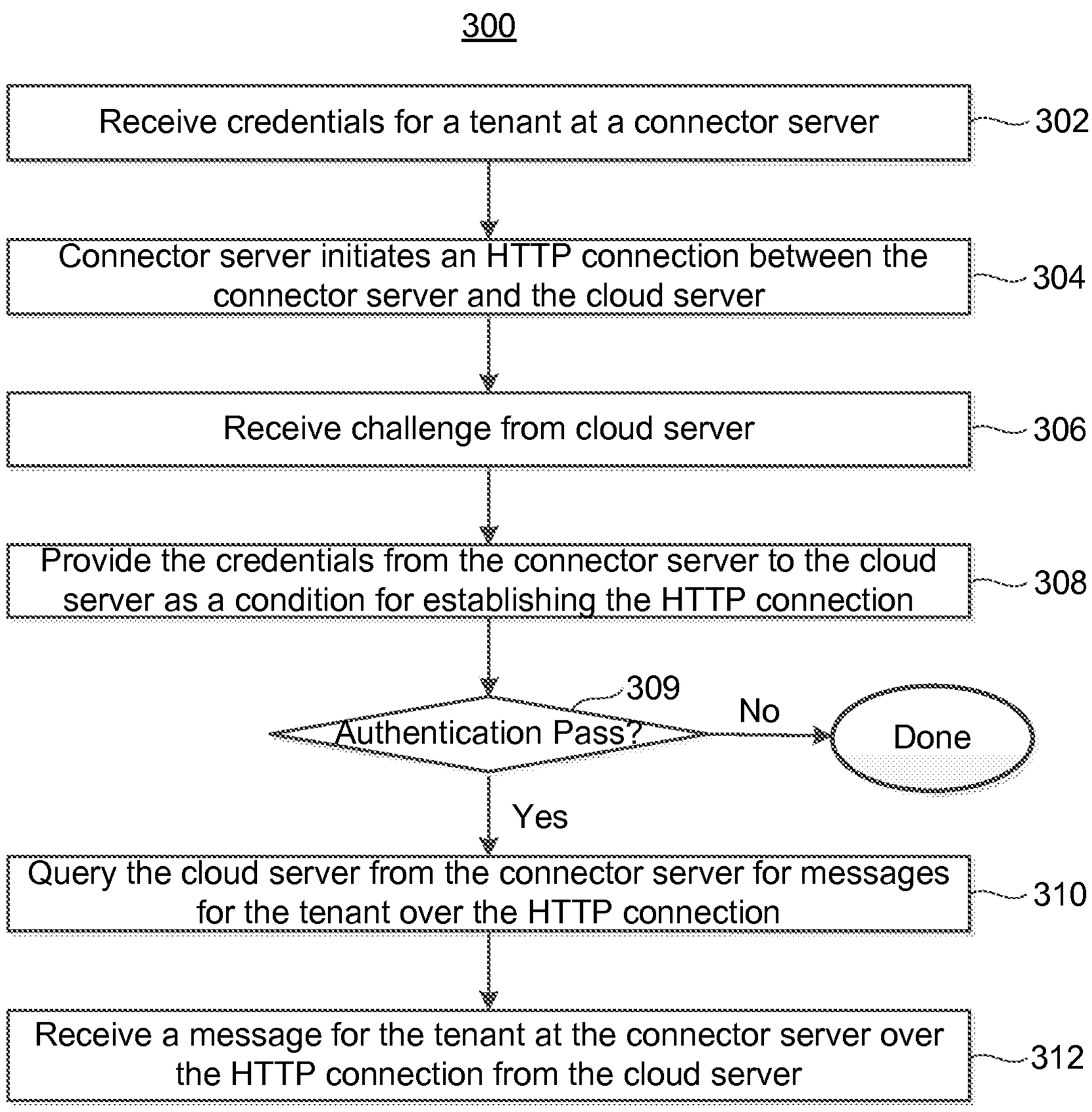

300
Receive credentials for a tenant at a connector server
302
Connector server initiates an HTTP connection between the connector server and the cloud server
304
Receive challenge from cloud server
306
Provide the credentials from the connector server to the cloud server as a condition for establishing the HTTP connection
308
309
Authentication Pass?
No
Done
Yes
Query the cloud server from the connector server for messages for the tenant over the HTTP connection
310
Receive a message for the tenant at the connector server over the HTTP connection from the cloud server
312

Figure 3

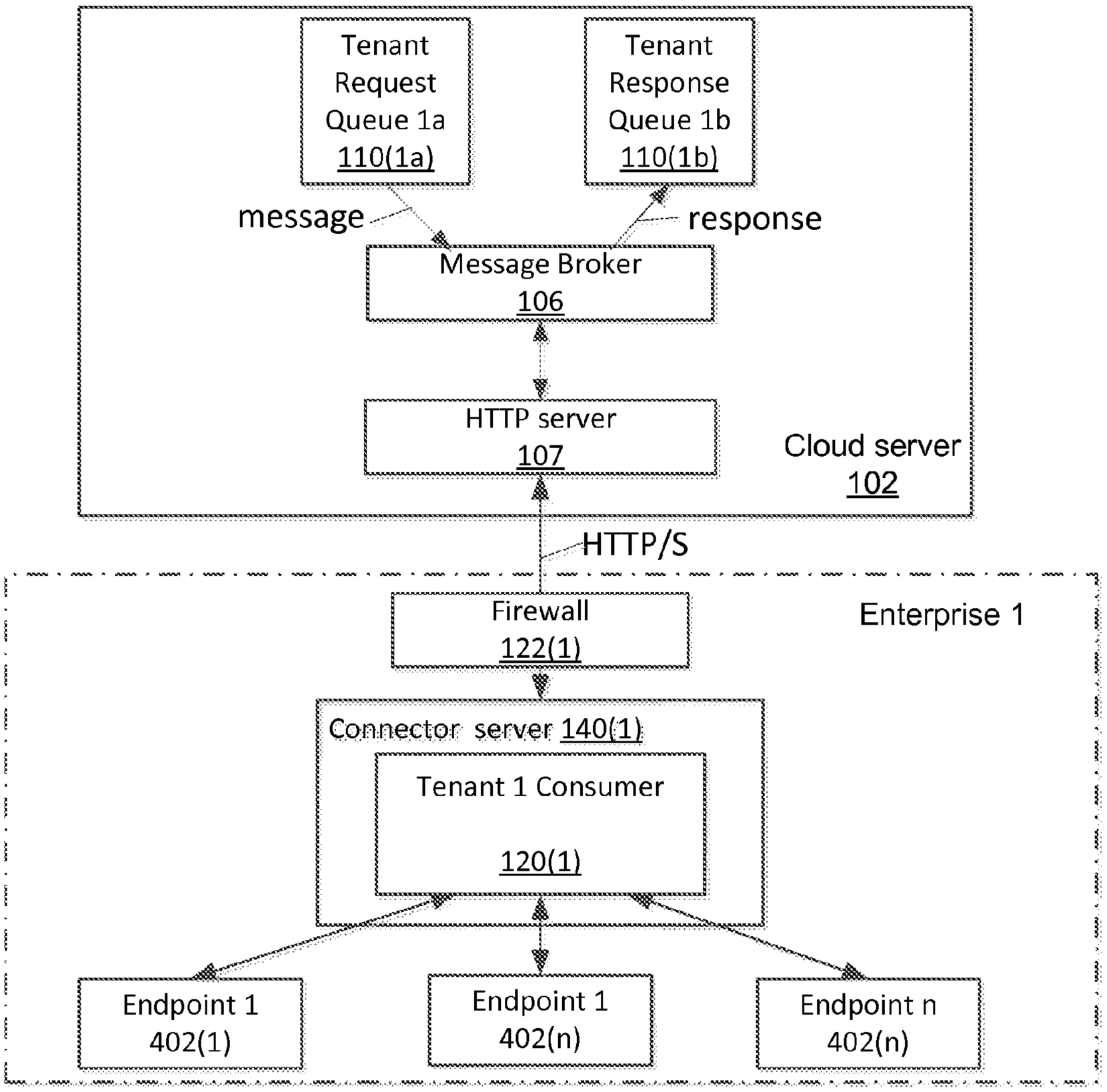


Figure 4A

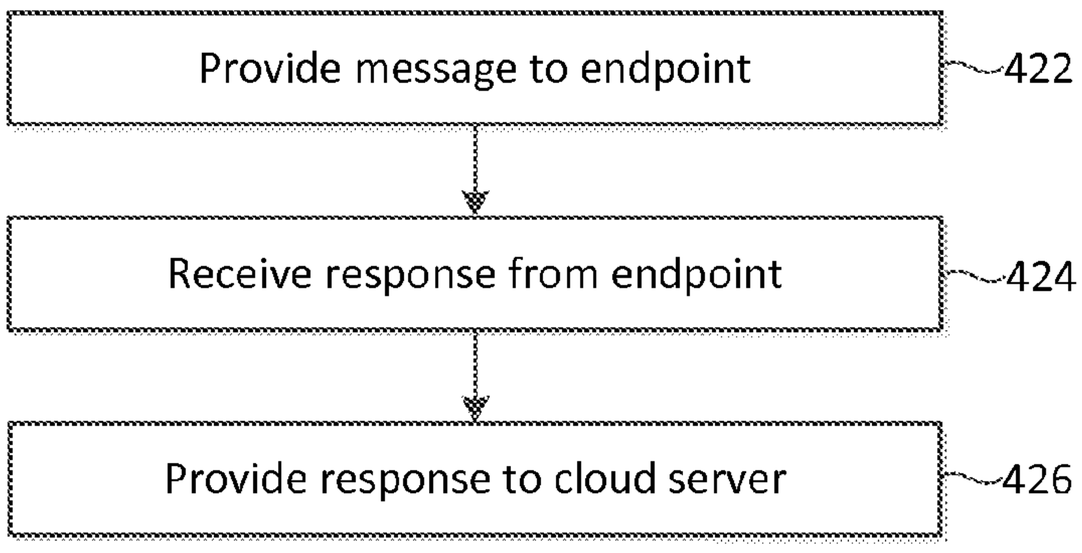


Figure 4B

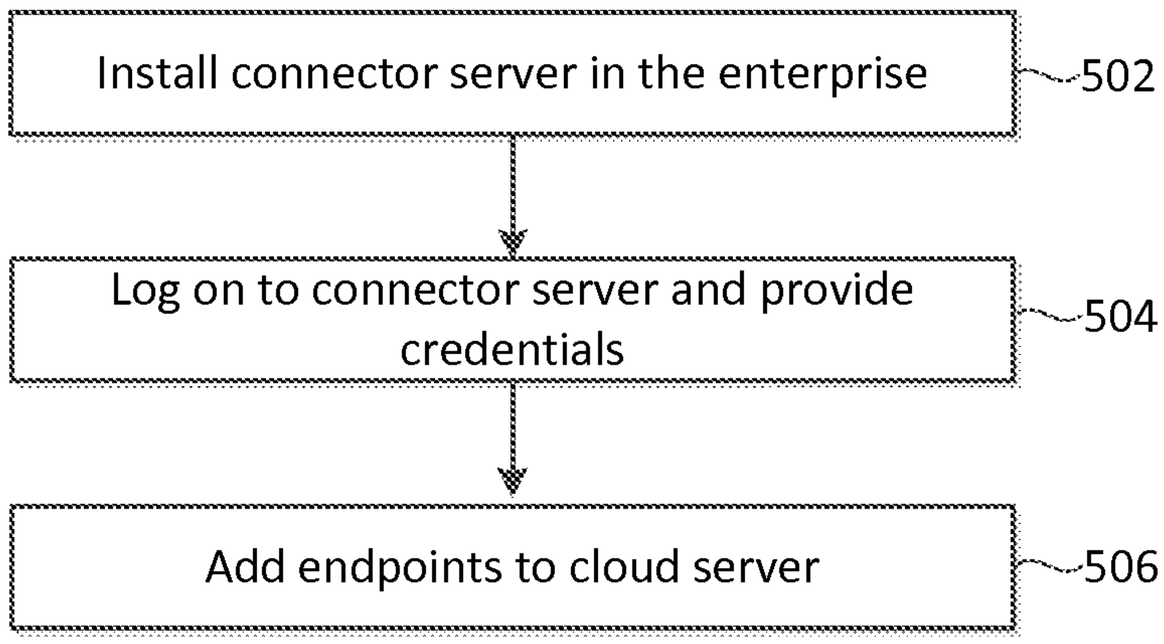


Figure 5

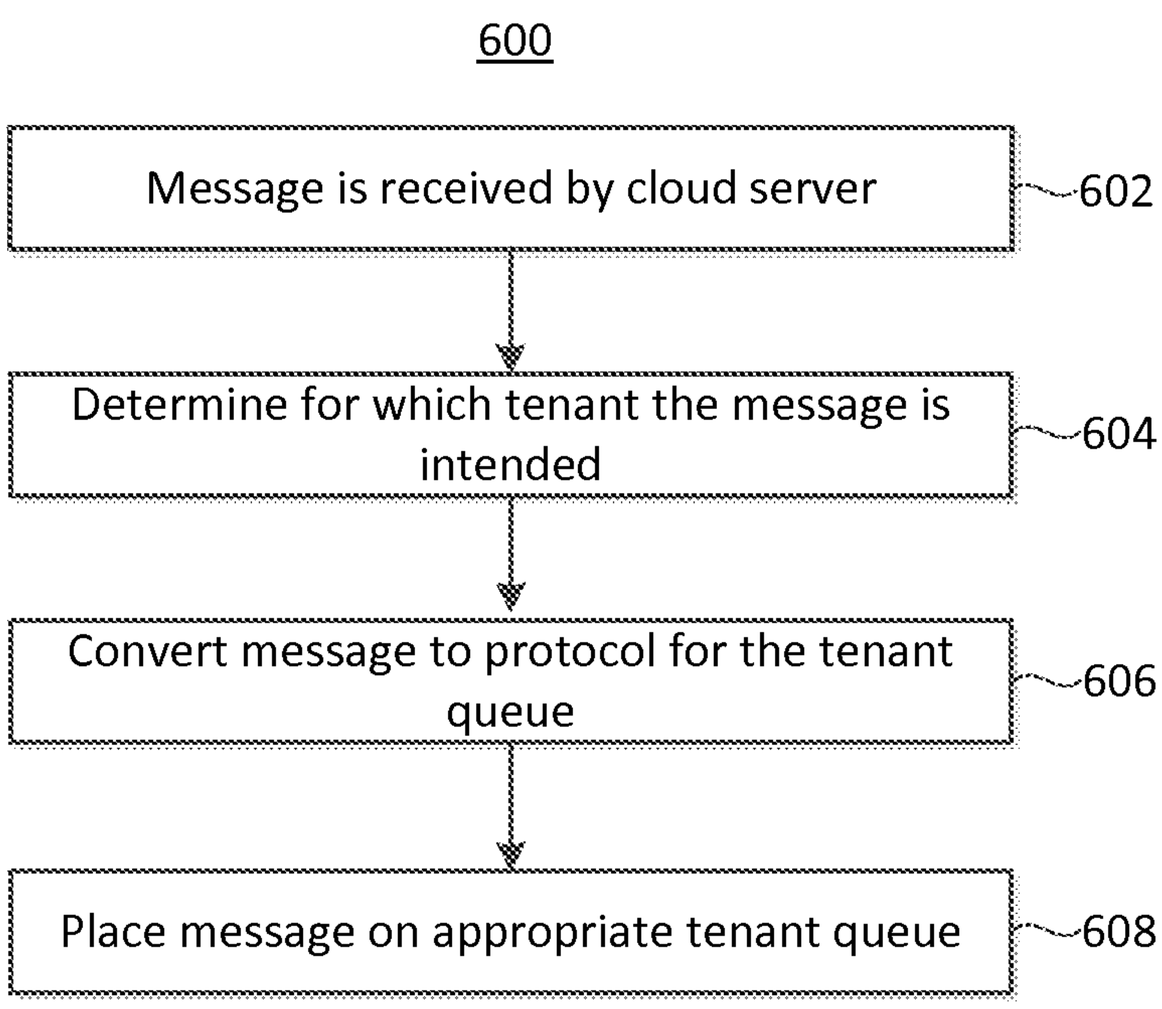


Figure 6

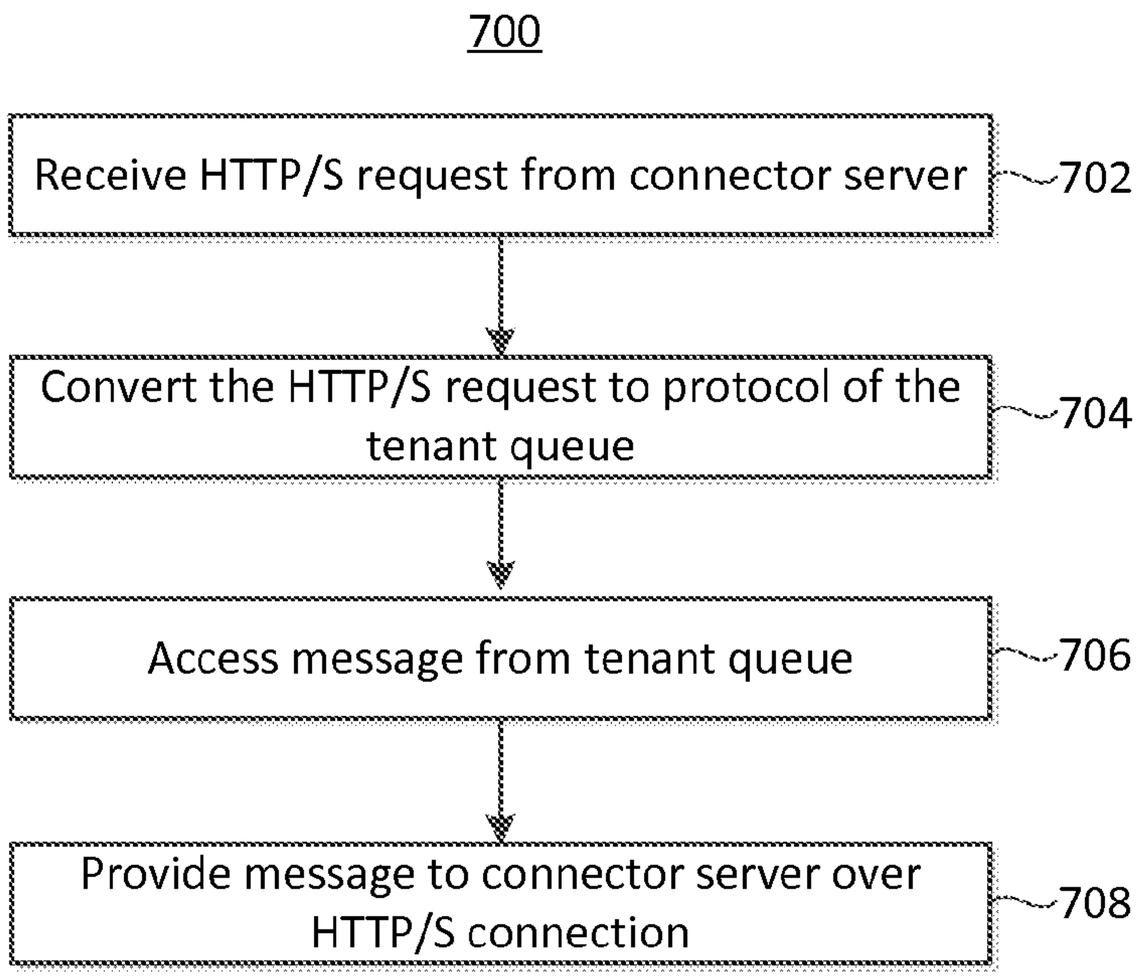


Figure 7

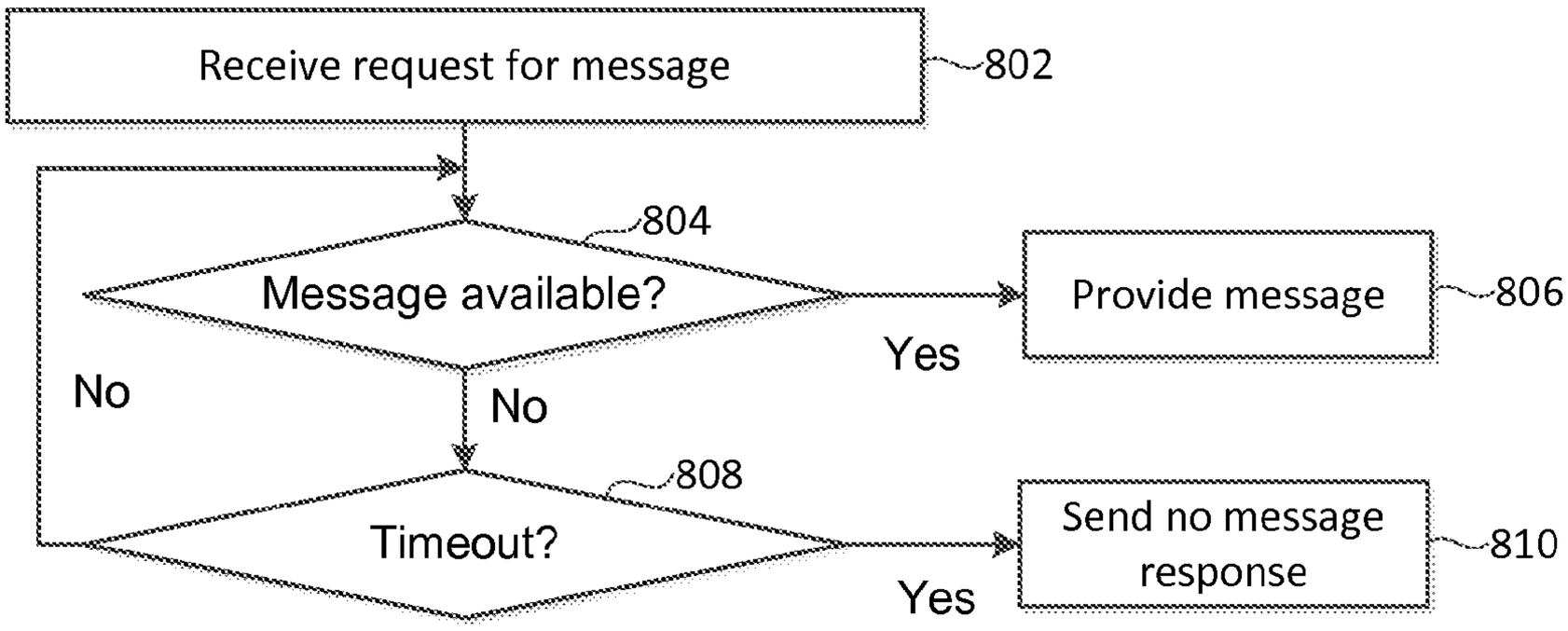


Figure 8

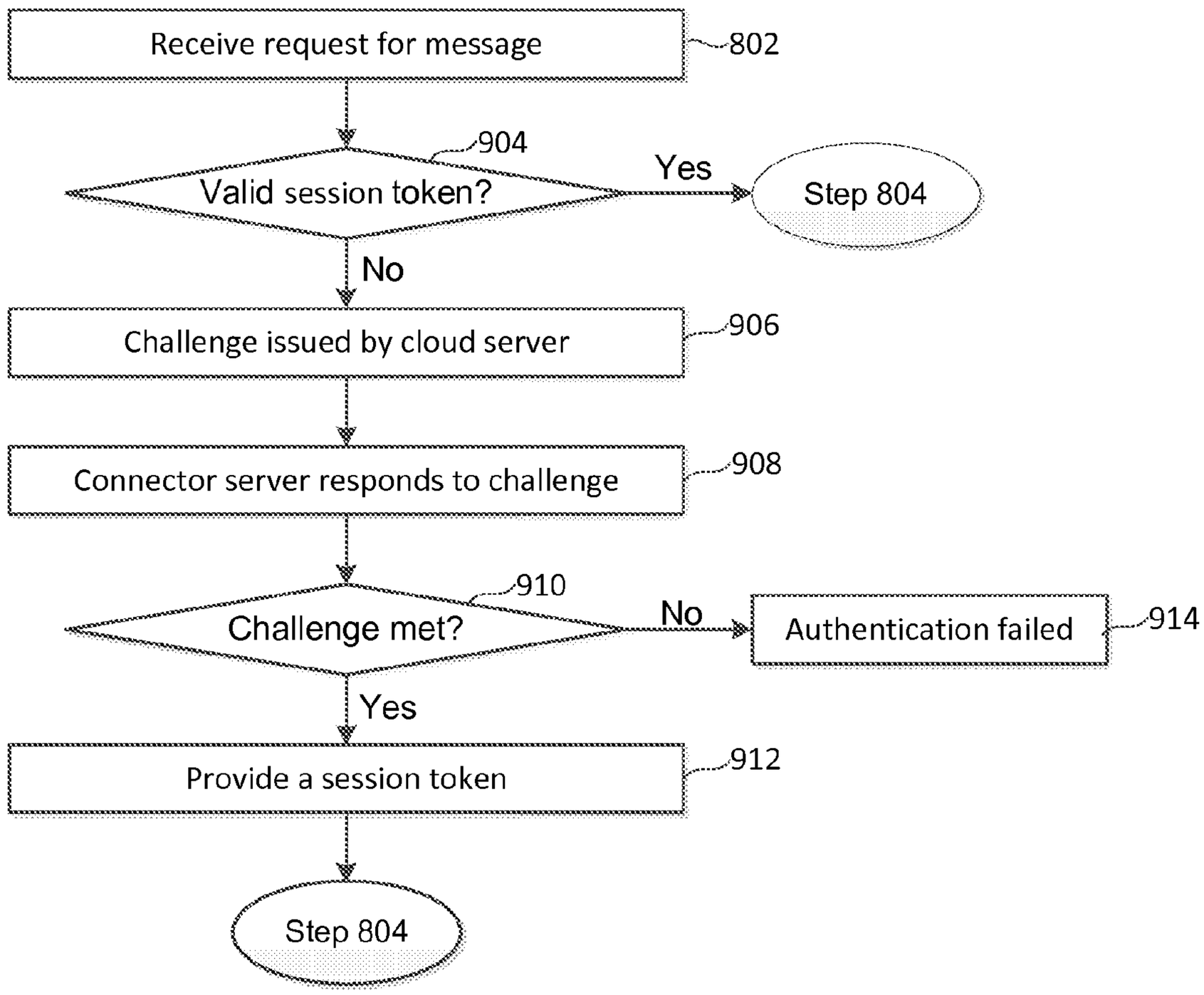

Receive request for message
802
904
Valid session token?
Yes
Step 804
No
Challenge issued by cloud server
906
Connector server responds to challenge
908
910
Challenge met?
No
Authentication failed
914
Yes
Provide a session token
912
Step 804

Figure 9

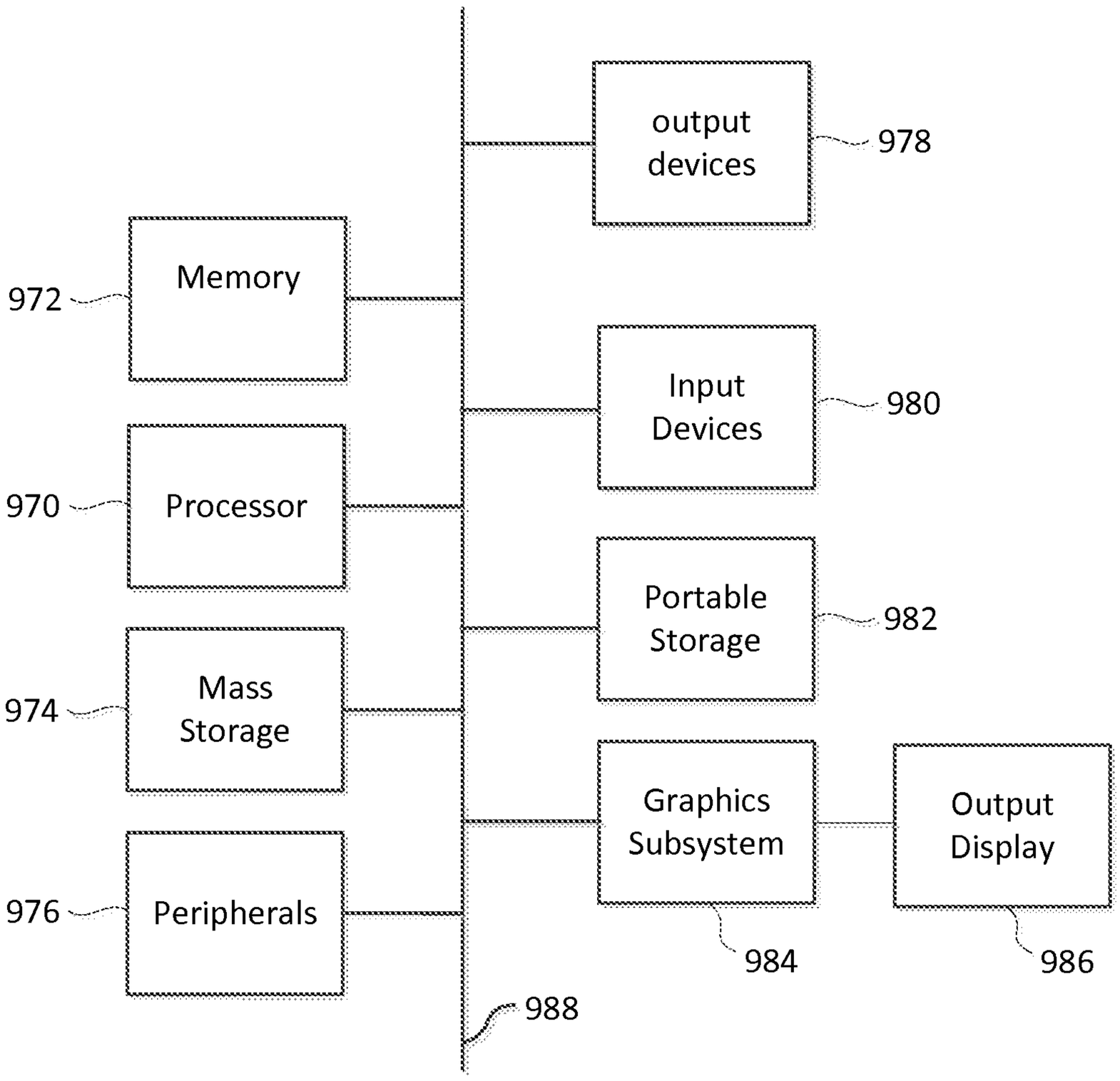

output devices
978
Memory
972
Input Devices
980
Processor
970
Portable Storage
982
Mass Storage
974
Graphics Subsystem
Output Display
Peripherals
976
984
986
988

Figure 10

SECURELY PROVIDING MESSAGES FROM THE CLOUD

BACKGROUND

The present disclosure relates to securely providing messages from a computing network.

Enterprises often need to allow messages to be delivered from outside of the enterprise. Typically enterprises have firewalls to provide security for their computers and associated resources. The firewalls, which are designed to provide security, can make it difficult for an external source to provide messages.

One possible solution is to open a port in the firewall to allow the messages to be sent from an external source into an enterprise. However, such a solution potentially leaves the enterprise vulnerable to attack.

Another possible solution is a dedicated Virtual Private Network (VPN) link from a server outside of the enterprise to a server within the enterprise. However, the VPN may present a high security risk as it may expose the entire enterprise to attack from the external server.

BRIEF SUMMARY

According to one embodiment of the present disclosure a message is received at a first server that provides a cloud service to a plurality of tenants. The message complies with a protocol other than HTTP. The message is put on a message queue for a first tenant of the plurality of tenants. An HTTP connection is established in response to a request from a second server to establish the HTTP connection between the second server and the first server. The first server receives an HTTP request from the second server over the HTTP connection for a message for the first tenant. The message is provided to the second server over the HTTP connection in response to determining that the second server is authorized to receive messages for the first tenant.

This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter. The claimed subject matter is not limited to implementations that solve any or all disadvantages noted in the Background.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** depicts one embodiment of an environment in which embodiments may be practiced.

FIG. **2** is a flowchart of one embodiment of a process of providing messages securely.

FIG. **3** is a flowchart of one embodiment of a process of a connector server receiving a message from a cloud server.

FIG. **4**A is a diagram showing one embodiment of endpoints in an enterprise.

FIG. **4**B is one embodiment of a process of a connector server processing the messages it receives.

FIG. **5** is a flowchart of one embodiment of configuring the connector server.

FIG. **6** is a flowchart of one embodiment of a process of processing messages received by the cloud server.

FIG. **7** is a flowchart of one embodiment of a process of a cloud server processing a request from a connector server for a message for a tenant.

FIG. **8** is a flowchart of one embodiment of a process of a cloud server looking for new messages for a tenant.

FIG. **9** is a flowchart of one embodiment of a process of further details of a cloud server responding to a request for a message.

FIG. **10** illustrates a high level block diagram of a computer system which can be used to implement the technology described herein.

DETAILED DESCRIPTION

Techniques for securely providing messages to an enterprise are described herein. These messages may be provided from a cloud server. Enterprises often protect their valuable computer related assets from attack over a computer network by installing a firewall or other similar device. However, enterprises would also like to allow communication from the outside to be received. Thus, a problem is how to allow messages to be received without compromising security.

One embodiment includes a cloud server that receives messages that are destined for various enterprises that may be tenants of the cloud server. The messages may be of a type that are not allowed through a firewall of the given enterprise, absent the enterprise taking special steps such as opening a port to allow the messages through. However, rather than having the cloud server push the messages to the enterprise, the cloud server holds the messages on various tenant message queues. The cloud server may have a tenant message queue for each tenant for which it provides cloud services. When a new message is received by the cloud server, the cloud server places the message on the appropriate tenant message queue.

To obtain a message for a tenant, the enterprise has a connector server that first initiates a HyperText Transfer Protocol (HTTP) connection with the cloud server, using tenant credentials that were previously provided. After the cloud server authenticates the connection server, the HTTP connection is established. This may be an HTTP/S (Hypertext Transfer Protocol Secure) connection. As the term is used herein, an HTTP connection includes an HTTP/S connection. Then, the connector server can send an HTTP request to the cloud server to pull messages into the enterprise. The connector server may establish the HTTP connection through a firewall associated with the enterprise. However, the enterprise need not open up ports in the firewall to obtain the messages. Therefore, the enterprise is not vulnerable to external attacks via the port. Further details are provided below.

As will be appreciated by one skilled in the art, aspects of the present disclosure may be illustrated and described herein in any of a number of patentable classes or context including any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof. Accordingly, aspects of the present disclosure may be implemented entirely hardware, entirely software (including firmware, resident software, micro-code, etc.) or combining software and hardware implementation that may all generally be referred to herein as a "circuit," "module," "component," or "system." Furthermore, aspects of the present disclosure may take the form of a computer program product embodied in one or more computer readable media having computer readable program code embodied thereon.

Any combination of one or more computer readable media may be utilized. The computer readable media may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, or semiconductor system, appara-

tus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an appropriate optical fiber with a repeater, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device. Program code embodied on a computer readable signal medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present disclosure may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Scala, Smalltalk, Eiffel, JADE, Emerald, C++, CII, VB.NET, Python or the like, conventional procedural programming languages, such as the "c" programming language, Visual Basic, Fortran 2003, Perl, COBOL 2002, PHP, ABAP, dynamic programming languages such as Python, Ruby and Groovy, or other programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider) or in a cloud computing environment or offered as a service such as a Software as a Service (SaaS).

Aspects of the present disclosure are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatuses (systems) and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable instruction execution apparatus, create a mechanism for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that when executed can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions when stored in the computer readable medium produce an article of manufacture including instructions which when executed, cause a computer to implement the function/act specified in the flowchart and/or block diagram block or blocks. The computer program instructions may also be loaded onto a computer, other programmable instruction execution apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatuses or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

FIG. **1** depicts one embodiment of an environment in which embodiments may be practiced. The environment includes a cloud server **102** and several enterprises (Enterprise **1** . . . Enterprise n). The enterprises may be tenants of the cloud server **102**. The cloud server **102** may provide cloud services **119** to the enterprises. In one embodiment, one of the cloud service **119**(**1**) is identity management. Identity management may pertain to the management of individual identities, their authentication, authorization, roles, and privileges within or across system and enterprise boundaries. "Identity management" may also be referred to as "access and identity management" (or AIM). Other cloud services **119**(**2**)-**119**(*n*) may be provided.

The cloud server **102** provides messages to the enterprises. The message delivery could be in connection to one of the cloud services **119**. In one embodiment, the messages are part of the identity management service **119**(**1**). For example, these could be requests made by identity management business logic.

To provide the messages, the cloud server **102** has a message server **104**, message broker **106**, HTTP server **107**, tenant keys **111**, and tenant queues **110**(**1**)-**110**(*n*). The tenant queues **110**(**1**)-**110**(*n*) are used to store messages that are destined for the various tenants. Each tenant that is served by the cloud server **102** may have its own tenant queue. In one embodiment, the tenant queues **110**(**1**) are JMS (Java Message Service) queues. In one embodiment, cloud server **102** uses ActiveMQ™ as a message broker to implement the tenant queues **110**(**1**)-**110**(*n*). In one embodiment, the message broker **106** creates an instance of a tenant queue for each tenant that is served by the cloud server **102**.

The message server **104** is able to receive messages from network **107**. The network **107** could be any network including, but not limited to, the Internet, a local area network (LAN), and/or a wide area network (WAN). These messages may be destined for the various tenants hosted by the cloud server **102**. Also, these messages may be associated with one of the cloud services **119**. In one embodiment, these messages have any format except HTTP. Rather than push the messages to the appropriate connector server **140**, the message server **104** puts the messages onto the appropriate tenant queue **110**(**1**)-**110**(*n*). In one embodiment, the message server **104** is a Lightweight Directory Access Protocol (LDAP) server. However, the message server **104** is not limited to being an LDAP server.

FIG. **1** depicts several enterprises (Enterprises **1** . . . n), which may be different tenants of the cloud server **102**. Each enterprise may be associated with a different enterprise such as a business, organization, etc. Each enterprise has a connec-

tor server **140**(**1**)-**140**(*n*), which may have a firewall **122**(**1**)-**120**(*n*) associated with it. The connector server **140** communicates with the cloud server **102** to obtain cloud services, obtain messages, etc.

Each connector server **140**(**1**)-**140**(*n*) may have a consumer **120**(**1**)-**120**(*n*). The consumer **120**(**1**)-**120**(*n*) may be a consumer of messages that were stored on the respective tenant queue **110**(**1**)-**110**(*n*). The consumer **120** may provide these messages to various endpoints in the respective enterprise (not depicted in FIG. **1**). An endpoint may be a final target of the messages. Examples of endpoints include, but are not limited to, active directories, databases, and mainframes.

Thus, note that the connector server **140** pulls the messages into the enterprise. A given consumer **120** may be configured with credentials associated with its associated tenant, such that it is able to obtain messages. To obtain messages, the consumer may initiate an HTTP connection with the cloud server **102** through the firewall **122** associated with the connector server **140**. The HTTP connection may be established over network **109**. This may be an HTTP/S connection.

The firewall **122** may be a hardware firewall, a software firewall, or some combination of hardware and software. There may be more than one firewall associated with a given connector server **140**. In one embodiment, there is both a hardware firewall and a software firewall associated with the connector server **140** in a given enterprise. For example, there may be a software firewall that runs on the connector server **140**, as well as a hardware firewall outside of the connector server **140**.

In one embodiment, the firewall **122** is a stand-alone device that is separate from the connector server **140**. As one example, the firewall **122** could be implemented within a router. The firewall **122** could use packet filtering to examine the header of a packet that is incoming from network **109** to determine its source and destination. This information may be compared to a set of predefined or user-created rules that determine whether the packet is to be forwarded or dropped. In one embodiment the firewall **122** is configured such that it would drop a packet if the packet is in the format it had when received by the cloud server **102**. For example, the firewall **122** may be configured to drop a packet sent by an LDAP server. Thus, the cloud server **102** would not be able to simply send the message to the connector server **140**.

As noted, the firewall **122** may be configured such that it would not allow the messages to be passed to the connector server **140** in their "raw" form in which they are received by the message server **104** in the cloud server **102**. One possible option would be to configure the firewall **122** such that it could receive the messages in the "raw" form. However, such a change to the firewall **122** could make the enterprise vulnerable to attacks over the network **109**.

The network **109** can include, e.g., the Internet, another wide area network, and/or a local area network. Networks **107** and **109** may be the same or different networks. Also, the various connector servers **140**(**1**)-**140**(*n*) may access the cloud server **102** over the same or different networks.

An enterprise may have an HTTP proxy server **145** between the connector server **140** and the network **109**. Such an HTTP proxy server **145** is depicted in enterprise "n." An HTTP proxy server **145** may take HTTP requests from the connector server **140** and forward them to the network **109**. The HTTP proxy server **145** may apply access policies to control/block access to sites that are prohibited by the enterprise. The HTTP proxy server **145** may have other purposes, as well. In one embodiment, a tenant consumer **120** is configured with authentication credentials such that it can authenticate with the HTTP proxy server **145**. Therefore, the connector server **140** is able to establish an HTTP connection with the cloud server **102** via the HTTP proxy server **145**. As depicted in FIG. **1**, the HTTP connection is between the cloud server **102** and the HTTP proxy server **145**.

The cloud server **102** is able to field requests from the connector servers **140**(**1**)-**140**(*n*), determine whether the connector server **140** is entitled to messages from a given tenant queue **110**, and, if so, provide the messages. In one embodiment, the tenant keys **111** are used to determine whether access should be granted. The tenant keys **111** may include, but are not limited to, user name, password, and tenant name. In one embodiment, the HTTP server **107** receives HTTP requests. These may be HTTP/S requests. These HTTP requests may be forwarded to the message broker **106**.

In one embodiment, the message broker **106** is able to convert an HTTP request to a request that is compatible with the tenant queue **110**. In one embodiment, the message broker **106** converts an HTTP request to a request to obtain a message from a JMS queue. In one embodiment, the message broker **106** is an ActiveMQ™ message broker.

FIG. **2** is a flowchart of one embodiment of a process **200** of providing messages securely. The process **200** may be practiced in the environment of FIG. **1**, but is not so limited. In one embodiment, process **200** is implemented by a cloud server **102**. In one embodiment, the cloud server **102** has a processor that is configured to implement functionality described in process **200**. Process **200** describes processing one message for one tenant that is received by the cloud server **102**. Typically, the cloud server **102** processes many messages for each of numerous tenants.

In step **202**, the cloud server **102** receives a message having a protocol other than HTTP. In one embodiment, the message is received by an LDAP server. The message server **104** in the cloud server may receive the message from one of the cloud services **119**. In one embodiment, the message is from the identity management service **119**.

In step **204**, the cloud server **102** puts the message on a message queue for a tenant to whom the message is intended. For example, the message is placed onto one of the tenant queues **110**(**1**)-**110**(*n*). In one embodiment, step **204** includes the cloud server **102** identifying a distinguished name in the message in order to determine which tenant queue **110** to place the message on. Further details are discussed in connection with FIG. **6**.

In step **206**, the cloud server **102** receives a request from a server in an enterprise to open an HTTP connection between the cloud server **102** and the server in the enterprise. The request could come from the connector server **140**. The request may come from a HTTP proxy server **145**, on behalf of the connector server **140**.

In step **208**, the cloud server **102** determines whether the server is entitled to receive messages for the first tenant. In one embodiment, the cloud server **102** determines whether the connector server **140** is authorized to receive the tenant message. Of course, in some cases, the cloud server **102** may determine that a connector server **140** is not entitled to receive messages for a particular tenant. In that case (step **209**=no), the process **200** concludes.

In step **210**, an HTTP connection is established between the cloud server **102** and the server that requested the connection. In one embodiment, an HTTP connection is established between the cloud server **102** and the connector server **140** in response to the cloud server **102** validating the connector server **140** (step **209**=yes). In one embodiment, the HTTP connection is between the cloud server **102** and a proxy server

145 between the connector server 140 and cloud server 102. In one embodiment, the HTTP connection is an HTTP/S connection.

In one embodiment, step 210 includes providing a token (e.g., session token) to the connector server 140. The connector server 140 may use the token throughout a session with the cloud server 102. The session will remain valid for a period of time dictated by the HTTP server 107.

In step 212, the cloud server 102 receives an HTTP request from the connector server 140 for a message for the tenant. Step 212 may include receiving the request over the HTTP connection that was established in step 210. The HTTP request may be passed to the cloud server 102 from the proxy server 145. In one embodiment, the connector server 140 provides the token. FIG. 9 provides further details for one embodiment of determining whether a connector server 140 is entitled to receive messages for a tenant of the cloud server 102.

Step 214 includes the cloud server 102 providing the message to the connector server 140 over the HTTP connection in response to validating the connector server 140. Note that the cloud server 102 may validate the connector server 140 in response to the request to open the HTTP connection and/or the request for the message. Step 214 may include the message being sent through a firewall 122 associated with the connector server 140.

FIG. 3 is a flowchart of one embodiment of a process 300 of a connector server 140 receiving a message from a cloud server 102. The process 300 may be practiced in the environment of FIG. 1. In one embodiment, a connector server 140 implements process 300.

Step 302 includes the connector server 140 receiving credentials for a tenant. The credentials may be provided from the cloud server 102, in one embodiment. In one embodiment, step 302 includes configuring the connector server 140 with the credentials. This may be achieved by a system administrator entering in credentials that were provided by the cloud server 102 into a user interface at the connector server 140. Thus, in one embodiment, step 302 includes the connector server 140 receiving the credentials through a user interface.

The credentials may include, but are not limited to a tenant name, user name, and password. In one embodiment, step 302 includes configuring the connector server 140 with a URL associated with the cloud server 102. In one embodiment, this is the URL of HTTP server 107. Step 302 may include providing the various credentials and URL to the connector server 140.

Step 304 includes the connector server 140 initiating an HTTP connection between the connector server 140 and the cloud server 102. In one embodiment, the connector server 140 sends a request to the cloud server 102 to open an HTTP connection. In one embodiment, the request to open the connection is sent from the proxy server 145.

The connector server 140 may initiate the HTTP connection through a firewall 122 associated with the connector server 140. In one embodiment, the firewall 122 is configured to allow HTTP connections to be established, providing that they are initiated from within the enterprise. However, the firewall 122 may be configured to prevent an HTTP connection from being established if the initiator is from outside of the enterprise. In one embodiment, step 304 includes the connector server 140 using the URL that it was provided in step 302 to contact the cloud server 102. Step 304 may include sending a request to that URL.

In one embodiment, the HTTP connection is a persistent connection. A persistent connection allows multiple requests to be sent from the connector server 140 to the cloud server 102 while the HTTP connection remains open. However, the HTTP connection is not required to be a persistent connection. In one embodiment, the connector server 140 opens a new HTTP connection each time a request for a message is to be sent to the cloud server 102.

As noted in the discussion of FIG. 1, the enterprise may have a proxy server 145 between the connector server 140 and the network 109. In this case, step 304 may include the HTTP proxy server 145 intercepting the request to open the HTTP connection. The HTTP proxy server 145 may send a challenge to the connector server 140 when the connector server 140 attempts to open the connection to the cloud server 102. The connector server 140 then responds to the challenge with credentials that have been previously provided. Upon passing the challenge, the HTTP proxy server 145 may then proceed to forward the request to open the connection to the cloud server 102.

Step 306 includes the connector server 140 receiving a challenge from the cloud server 102 in response to the request to open the HTTP connection.

Step 308 includes the connector server 140 providing the tenant credentials to the cloud server 102 in response to the challenge as a condition for establishing the HTTP connection.

If the connector server 140 is unable to provide the proper credentials, the process 300 ends (step 309=no). If the connector server 140 successfully responds to the challenge (step 309=yes), then the HTTP connection is established.

Step 310 includes the connector server 140 querying the cloud server 102 for messages for the tenant over the HTTP connection. The connector server 140 may query the cloud server 102 using the HTTP connection through the firewall 122 associated with the connector server 140.

Step 312 includes the connector server 140 receiving a message for the tenant from the cloud server 102 over the HTTP connection. The message may be received through a firewall 122 associated with the connector server 140.

After the connector server 140 receives a message it may provide it to an endpoint, in one embodiment. FIG. 4A is a diagram showing one embodiment of endpoints in an enterprise. The connector server 140 has connections to a number of endpoints 402(1)-402(n) within the enterprise. The endpoints 402 may be final consumers of the messages. An example of an endpoint is a database. Another example of an endpoint is an LDAP server. Another example of an endpoint is a main frame computing system. The endpoints 402 may be involved with the cloud services 119. In one embodiment, messages are sent to the end points 402, as a part of providing the cloud service. For example, the identity management service 119(1) may send a message to one of the endpoints 402. FIG. 4A will be discussed in more detail when discussing FIG. 4B.

FIG. 4B is one embodiment of a process 400 of a connector server 140 processing the messages it receives. As noted previously, the connector server 140 establishes a HTTP connection with the cloud server 102. In FIG. 4B, the tenant queue is depicted as having a tenant request queue 110(1a) and a tenant response queue 110(1b). The message broker 106 consumes a message from the tenant request queue 110 (1a) and provides it to the connector server 140 over the HTTP/S connection. In step 422, the connector server 140 provides a message to an endpoint 402. An example of a message is a request to create an account on an LDAP server. Another example of a message is a request to modify an account on a mainframe.

In step 424, the connector server 140 receives a response to the message form the endpoint 402. In step 426, the connector

server **140** provides the response to the cloud server **102**. In one embodiment, the message broker **106** places the response on the tenant response queue **110**(**1***b*). The responses on the tenant response queue **110**(**1***b*) may be processed by the message server (**104**, FIG. **1**). For example, the message server **104** may return the responses to the source that provided the message to the cloud server **102**.

Prior to being able to communicate with the cloud server **102**, the connector server **140** may be configured. FIG. **5** is a flowchart of one embodiment of configuring the connector server **140**. FIG. **5** may be performed after receiving the tenant credentials (e.g., step **302**, FIG. **3**). Prior to the process of FIG. **5**, a tenant consumer **120** may be accessed. The tenant consumer **120** may be software to be installed on the connector server **140** to implement a connection server. In one embodiment, the tenant consumer **120** is accessed as a download from the cloud server **102**, or another server. In one embodiment, the tenant consumer **120** is accessed by obtaining computer readable storage, such as a CD-ROM.

In step **502**, a tenant consumer **120** is installed on the connector server **140**. In step **504**, the connector server **140** is logged onto and credentials are provided. The provided credentials are input to the tenant consumer **120** by the connector server **140** so that the tenant consumer **120** can use those credentials when challenged. For example, the user starts to run the connector server **140** and accesses a login screen. The user then provides various tenant credentials, which are input to the tenant consumer **120** by the connector server **140**. The URL of the cloud server **102** may also be provided at this time.

In step **506**, endpoints **402** may be added to the cloud server **102**. Step **506** may include logging on to a user interface provided by the cloud server **102** and providing information about endpoints **402** for which messages should be provided. This may include identifying the connector server **140** (and/or tenant consumer **120**). After the connector server **140** has been identified, the endpoints **402** associated with the connector server **140** may be provided to the cloud server **102**. For example, an Active Directory endpoint may be added in this manner. As other examples, a mainframe endpoint or a database endpoint may be added.

FIG. **6** is a flowchart of one embodiment of a process **600** of processing messages received by the cloud server **102**. Process **600** provides further details for step **204** or FIG. **2**. This process may be performed by the message server **104** in the cloud server **102**. In step **602**, a message is received by the message server **104** in the cloud server **102**. In one embodiment, the message server is an LDAP server. However, the message server **104** could be compliant with some other protocol. In one embodiment, the message is any protocol but an HTTP message. The message may be one that would not be able to be pass through the firewall **122** of the enterprise without configuring the firewall **122** in a way that would expose the enterprise to external attack.

In step **604**, the message server **104** determines which tenant the message is intended for. In one embodiment, the message server **104** examines the message to determine a distinguished name. A distinguished name is a unique name for an entry in a directory service. The distinguished name may comprise a hierarchy that provides a path to the tenant consumer **120**. In one embodiment, the distinguished name pertains to an object that is attempted to be accessed. This object may be associated with one of the endpoints **402**(**1**)-**402**(*n*). In one embodiment, the tenant can be uniquely determined based on the distinguished name. In one embodiment, the distinguished name contains information that directly identifies the tenant. In one embodiment, a server that provides the messages to the cloud server **102** has the task of providing information in the message that will uniquely provide the tenant.

In step **606**, the message is converted to a protocol of the tenant queue **110**. In one embodiment, the message is converted to a JMS message. In one embodiment, the message is converted into an ActiveMQ™ message.

In step **608**, the message is placed onto the appropriate tenant queue **110**(**1**)-**120**(*n*), as determined in step **604**.

FIG. **7** is a flowchart of one embodiment of a process of a cloud server **102** processing a request from a connector server **140** for a message for a tenant. Step **702** includes receiving an HTTP/S request at the cloud server **102** from the connector server **140**. Step **702** may be performed by HTTP server **107** in the cloud server **102**. The request might come from a proxy server **145**. The request might come from connector server **140**.

In step **704**, the cloud server **102** converts the HTTP/S request to a format that complies with a format for the tenant queue **110**. Step **704** might be performed by either the HTTP server **107**, the message broker **106**, or some other component. In one embodiment, the HTTP server **107** passes the HTTP/S request to the message broker **106**, which converts the HTTP/S request. In one embodiment, the HTTP/S request is converted to an ActiveMQ™ request. In one embodiment, the HTTP/S request is converted to a JMS request.

In step **706**, the message broker **106** accesses the message from the tenant queue **110**. In one embodiment, a JMS message is accessed. In step **708**, the message is provided to the connector server **140** over the HTTP/S connection.

Since the messages are not pushed to the connector server **140**, the connector server **140** needs to have some way of determining when messages are available. In one embodiment, the connector server **140** performs a type of efficient polling to look for new messages. In one embodiment, the connector server **140** sends a request for messages and waits for a response. Either a message is received or a no message available is received. Upon receiving a message, the connector server **140** may request another message. Upon receiving a no message is available response, the connector server **140** may submit another request for a message. FIG. **8** is a flowchart of a process from the perspective of the cloud server to further illustrate.

FIG. **8** is a flowchart of one embodiment of a process of a connector server **140** looking for new messages for a tenant. Prior to this process, an HTTP connection has been successfully opened. In step **802**, the cloud server **102** receives a request using the established the HTTP connection for a message from the tenant queue. Note that in one embodiment the connector server **140** has already been authenticated at this point.

In step **804**, the cloud server **102** determines whether a message is presently available on the tenant queue **110**. If so, the message is provided to the connector server **140** over the HTTP connection, in step **806**.

In there is presently not a message on the tenant queue, then the cloud server **102** may wait some period of time to see whether any messages become available. The period of time could be configurable. So long as there is not a timeout (step **808**=no), the cloud server **102** continues to wait for messages. If there is a timeout (step **808**=yes), the cloud server **102** replies with a no message response to the connector server **140** over the HTTP connection.

Thus, after sending a request for a message, the connector server **140** may simply wait until it either receives a message or receives a timeout message.

In one embodiment, the connector server **140** synchronizes its clock with a clock of the cloud server. This may help to prevent a different type of timeout issue. Since the messages are being put on tenant queues in the cloud server, they may have a time to live (or other time parameter) that is based on a clock of the cloud server **102**. When the connector server **140** accesses a message from the tenant queue, it may check the time parameter to determine whether the message has expired. If so, it may determine that the message should not be delivered to, for example, an endpoint **402**. When making this check, the connector server **140** may use its own clock. Thus, if its clock were not synchronized to the cloud server's clock, the connector server **140** could potentially mistakenly determine that a message has timed out. Therefore, in one embodiment, the connector server **140** synchronizes its clock to the clock of the cloud server **102**, or the clock upon which time to live for messages on the tenant queue are based.

The clock synchronization may be achieved by the connector server **140** sending a request to the cloud server **102** for the present time of the cloud server's clock. The synchronization is not required to be extremely precise. In one embodiment, there may be a few seconds difference between the clocks. Some messages may have a time to live that is much larger than this. In general, the precision of synchronization may depend on how long messages have to live.

In one embodiment, when the connector server **140** opens an HTTP connection, the cloud server **102** starts a session and provides a session token to the connector server **140**. FIG. **9** is a flowchart of one embodiment of a process of further details of a cloud server **102** responding to a request for a message. The process starts with a request from a connector server **140** for a message. This request may include a token that was provided to the connector server **140** when the connector server **140** was first authenticated. This was discussed in step **212** of FIG. **2**.

In step **904**, the cloud server **102** determines whether the session token is valid. If the connector server **140** provides a valid session token, then control passes to step **804** of FIG. **8**. Recall that step **804** is test to determine whether there are any messages on the tenant queue.

If the connector server **140** did not present a valid session token, then the cloud server **102** presents a challenge, in step **906**. The connector server responds to the challenge in step **908**. Step **906** and **908** may be similar to steps **306** and **308** from FIG. **3**, but from the perspective of the cloud server **102**. If the challenge is not met, then the cloud server **102** sends a message to the connector server **140** that authentication failed, in step **914**.

If the challenge was met (step **910**=yes), the cloud server **102** provides a session token to the connector server **140**, in step **912**. Then, control passes to step **804**.

FIG. **10** illustrates a high level block diagram of a computer system which can be used to implement the technology described herein. The computer system could be used to implement cloud server **102**, connector server **140**, and/or proxy server. In some cases, the computer system could be used to implement a firewall **122**. In some cases, multiple computer systems are used to implement one of the foregoing.

The computer system of FIG. **10** includes a processor unit **970** in communication with main memory **972**. Processor unit **970** may contain a single microprocessor, or may contain a plurality of microprocessors for configuring the computer system as a multi-processor system. These one or more processors can perform the methods described above. Main memory **972** stores, in part, instructions and data for execution by processor unit **970**. If the system described herein is wholly or partially implemented in software, main memory **972** can store the executable code when in operation. Main memory **972** may include banks of dynamic random access memory (DRAM) as well as high speed cache memory. For example, main memory **972** can store the tenant queues **110**, HTTP server **107**, message broker **106**, tenant keys **111**, tenant consumer **120**, firewall **122**, etc.

The system of FIG. **10** further includes a mass storage device **974**, peripheral device(s) **976**, user input device(s) **980**, output devices **978**, portable storage medium drive(s) **982**, a graphics subsystem **984** and an output display **986**. For purposes of simplicity, the components shown in FIG. **10** are depicted as being connected via a single bus **988**. However, the components may be connected through one or more data transport means. For example, processor unit **970** and main memory **972** may be connected via a local microprocessor bus, and the mass storage device **974**, peripheral device(s) **976**, portable storage medium drive(s) **982**, and graphics subsystem **984** may be connected via one or more input/output (I/O) buses. Mass storage device **974**, which may be implemented with a magnetic disk drive or an optical disk drive, is a non-volatile storage device for storing data and instructions for use by processor unit **970**. In one embodiment, mass storage device **974** stores the system software for implementing the technology described herein for purposes of loading to main memory **972**. Peripheral device(s) **976** may include any type of computer support device, such as an input/output (I/O) interface, to add additional functionality to the computer system. For example, peripheral device(s) **976** may include a network interface for connecting the computer system to a network, a modem, a router, etc. User input device(s) **980** provides a portion of a user interface (e.g., to allow user to enter configuration information for configuring connector server **140**, cloud server **102**, proxy server **145**, firewall **122**, etc.). User input device(s) **980** may include an alpha-numeric keypad for inputting alpha-numeric and other information, or a pointing device, such as a mouse, a trackball, stylus, or cursor direction keys. In order to display textual and graphical information, the computer system of FIG. **10** includes graphics subsystem **984** and output display **986**. Output display **986** may include a cathode ray tube (CRT) display, liquid crystal display (LCD) or other suitable display device. Graphics subsystem **984** receives textual and graphical information, and processes the information for output to display **986**. Additionally, the system of FIG. **10** includes output devices **978**. Examples of suitable output devices include speakers, printers, network interfaces, monitors, etc.

The components contained in the computer system of FIG. **10** are those typically found in computer systems suitable for use with the technology described herein, and are intended to represent a broad category of such computer components that are well known in the art. Thus, the computer system of FIG. **10** can be a personal computer, mobile computing device, smart phone, tablet, workstation, server, minicomputer, mainframe computer, or any other computing device. The computer can also include different bus configurations, networked platforms, multi-processor platforms, etc. Various operating systems can be used.

One embodiment disclosed herein includes a system for securely providing messages. The system comprises a processor that is configured to provide a cloud service for a plurality of tenants, receive a message having a protocol other than HTTP, and put the message on a message queue for a first tenant of the plurality of tenants. The processor is further configured to receive a request from a requesting device to open an HTTP connection between the system and the requesting device. The processor is further configured

receive, from the requesting device, an HTTP request for a message for the first tenant. The processor is further configured to validate that the requesting device is authorized to receive messages for the first tenant. The processor is further configured provide the message to the requesting device over the HTTP connection in response to validating the requesting device.

One embodiment disclosed herein includes a computer program product comprising a computer readable storage medium comprising computer readable program code embodied therewith. The computer readable program code comprises computer readable program code configured to provide a cloud service for a plurality of tenants. The computer readable program code is configured to receive a message having a protocol other than HTTP. The computer readable program code is configured to put the message on a message queue for a first tenant of the plurality of tenants. The computer readable program code is configured to receive a request from a first server to open an HTTP connection between a second server and the first server. The computer readable program code is configured to receive, from the first server, an HTTP request for a message for the first tenant. The computer readable program code is configured to validate that the first server is authorized to receive messages for the first tenant. The computer readable program code is configured to provide the message to the first server over the HTTP connection in response to validating the first server.

One embodiment disclosed herein includes a computer program product comprising a computer readable storage medium comprising computer readable program code embodied therewith. The computer readable program code comprises computer readable program code configured to initiate, from a first server, an HTTP connection between the first server and a second server. The computer readable program code is configured to provide the credentials from the first server to the second server as a condition for establishing the HTTP connection. The computer readable program code is configured to query the second server from the first server for messages for the tenant over the HTTP connection. The computer readable program code is configured to receive a message for the tenant at the first server over the HTTP connection from the second server.

The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various aspects of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

The terminology used herein is for the purpose of describing particular aspects only and is not intended to be limiting of the disclosure. As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises" and/or "comprising," when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof.

The corresponding structures, materials, acts, and equivalents of any means or step plus function elements in the claims below are intended to include any disclosed structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of the present disclosure has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the disclosure in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the disclosure. The aspects of the disclosure herein were chosen and described in order to best explain the principles of the disclosure and the practical application, and to enable others of ordinary skill in the art to understand the disclosure with various modifications as are suited to the particular use contemplated.

What is claimed is:

1. A method of providing messages securely, the method comprising:

receiving a request message at a cloud server that provides a cloud service to a plurality of tenants, the message complying with a protocol other than HyperText Transfer Protocol (HTTP);

putting the request message on a Java Messaging Service (JMS) tenant request message queue for a first tenant of the plurality of tenants, wherein the JMS tenant request message queue resides on the cloud server;

establishing an HTTP connection between the cloud server and a connector server in response to a request from the connector server to establish the HTTP connection between the connector server and the cloud server, wherein the connector server is located within an enterprise that is associated with the first tenant;

receiving, at the cloud server from the connector server over the HTTP connection, an HTTP request for a message for the first tenant;

accessing the request message from the JMS tenant request message queue for the first tenant; and

providing the request message from the cloud server to the connector server over the HTTP connection in response to the cloud server determining that the connector server is authorized to receive messages for the first tenant.

2. The method of claim **1**, further comprising:

the cloud server challenging the connector server for credentials for the first tenant in response to the request from the connector server to establish the HTTP connection; and

the cloud server verifying credentials provided by the connector server in response to the challenge as a condition of establishing the HTTP connection.

3. The method of claim **1**, further comprising:

the cloud server challenging the connector server for credentials for the first tenant in response to the request from the connector server for a message for the first tenant; and

the cloud server verifying credentials provided by the connector server in response to the challenge as a condition of providing the message for the first tenant to the connector server.

4. The method of claim **1**, further comprising the cloud server converting the HTTP request to a request format that is

compliant with a protocol associated with the message queue in order to remove the message from the message queue.

5. The method of claim **4**, wherein the converting the HTTP request to a request format that is compliant with the message queue comprises converting the HTTP request to a JMS request.

6. The method of claim **1**, wherein the receiving a message at the cloud server that provides a cloud service a plurality of tenants comprises:

receiving a message having a protocol that is not allowed through a firewall associated with the connector server.

7. The method of claim **1**, wherein the receiving a message at the cloud server that provides a cloud service to a plurality of tenants comprises receiving the message at a Lightweight Directory Access Protocol (LDAP) server.

8. The method of claim **1**, wherein the HTTP connection is an HTTP/S connection.

9. The method of claim **1**, wherein the request message is a request made by an identity management service provided through the cloud service.

10. The method of claim **1**, further comprising:

receiving a response message at the cloud server from the connector server over the HTTP connection; and

placing the response message on a JMS tenant response message queue for the first tenant, wherein the JMS tenant response message queue resides on the cloud server.

11. The method of claim **1**, further comprising:

synchronizing a clock of the cloud server with a clock of the connector server;

establishing a time to live parameter for the request message based on the clock of the cloud server; and

determining by the connector server whether the request message has timed out based on the time to live parameter and the clock of the connector server after it has been synchronized to the clock of the cloud server.

12. A system for securely providing messages, the system comprising:

a processor that is configured to:

provide a cloud service for a plurality of tenants via a cloud server;

receive a request message having a protocol other than HyperText Transfer Protocol (HTTP);

put the request message on a Java Messaging Service (JMS) tenant request message queue for a first tenant of the plurality of tenants, wherein the JMS tenant request message queue resides on the cloud server;

receive a request at the cloud server from a connector server to open an HTTP connection between the cloud server and the connector server, wherein the connector server is located within an enterprise that is associated with the first tenant;

receive, at the cloud server from the connector server, an HTTP request for a message for the first tenant;

validate that the connector server is authorized to receive messages for the first tenant;

access the request message from the JMS tenant request message queue for the first tenant; and

provide the request message to the connector server over the HTTP connection in response to validating the connector server.

13. The system of claim **12**, wherein the processor is further configured to:

challenge the connector server for credentials for the first tenant in response to the request to open the HTTP connection; and

check credentials provided by the connector server in response to the challenge as a condition of providing the message.

14. The system of claim **12**, wherein the processor is further configured to:

challenge the connector server for credentials for the first tenant in response to the request from the connector server for a message for the first tenant; and

verifying credentials provided by the connector server in response to the challenge as a condition of providing the message for the first tenant to the connector server.

15. The system of claim **12**, wherein the processor being configured to receive the message comprises the processor being configured to receive a Lightweight Directory Access Protocol (LDAP) message, the processor is configured to convert the LDAP message to a Java Messaging Service (JMS) message.

16. The system of claim **12**, wherein the processor is further configured to convert the HTTP request to a request format that is compliant with the JMS tenant request message queue to remove the message from the message queue.

17. The system of claim **12**, wherein the processor is further configured to convert the HTTP request to an ActiveMQ request to remove the message from the JMS tenant request message queue.

18. The system of claim **12**, wherein the cloud service is an identity management service, the request message is from the identity management service.

19. A computer program product comprising:

a non-transitory computer readable storage medium comprising computer readable program code embodied therewith, the computer readable program code comprising:

computer readable program code configured to provide a cloud service for a plurality of tenants via a cloud server;

computer readable program code configured to receive a request message at the cloud server having a protocol other than HyperText Transfer Protocol (HTTP);

computer readable program code configured to put the request message on a Java Messaging Service (JMS) tenant request message queue for a first tenant of the plurality of tenants, wherein the JMS tenant request message queue resides on the cloud server;

computer readable program code configured to receive a request from a connector server to open an HTTP connection between the connector server and the cloud server wherein the connector server is located within an enterprise that is associated with the first tenant, wherein the connector server is a tenant consumer of the JMS tenant request message queue;

computer readable program code configured to receive, at the cloud server from the connector server, an HTTP request for a message for the first tenant;

computer readable program code configured to validate that the connector server is authorized to receive messages for the first tenant;

computer readable program code configured to access the request message from the JMS tenant request message queue for the first tenant; and

computer readable program code configured to provide the request message from the cloud server to the connector server over the HTTP connection in response to validating the connector server.

20. The computer program product of claim **19**, wherein the request message is an LDAP message, the computer readable program code is configured to convert the LDAP to a Java Messaging Service (JMS) message.

21. The computer program product of claim 19, wherein the computer readable program code is configured to convert the HTTP request to an ActiveMQ request to remove the request message from the JMS tenant request message queue.

* * * * *