CIPA ACCESS POINT 2.1

INSTALLATION AND USER GUIDE

Contents

Overview

The CIPA Access Point 2.1 allows your company to send to and receive documents from the PEPPOL network using the AS2 and/or START protocols for AP to AP transmissions, with all the benefits of the PEPPOL infrastructure.

Description

The CIPA AP is composed of a CIPA START server, an AS2 server and a dispatcher working as common interface to the client. The clients send SBDH documents to the dispatcher, which extracts the relevant information from it and using the PEPPOL infrastructure determines the receiver's characteristics. Then it forwards the message to the START or AS2 servers depending on what protocol the receiver understands.

The CIPA team decided to take Mendelson AS2 open source solution as its standard AS2 endpoint, and the current version of the disptacher integrates automatically with it. But the dispatcher has been designed in a neutral way allowing it to integrate with any AS2 endpoint implementation, by just implementing a few Java classes.

The dispatcher has been developed as a web application, and it can be deployed in the same web server than the AS2 and START endpoints, as well as in a different one. For simplicity,

and because it'll be the common case, this guide assumes the 3 components are deployed on the same web server.

Installation

For convenience, the 3 components of the CIPA AP 2.1 have been released already deployed on a Tomcat server ready for use. You can find the .zip here. You can unzip and place the resulting folder wherever you want in your file system.

Alternatively you can get the components' code from joinup and sourceforge, and compile the components yourself:

- dispatcher
- start-server
- mendelson server source code , mendelson server binaries

Please note that Mendelson is by default integrated with a Jetty web server, if you want the 3 components to run on the same web server we recommend you to take our released Tomcat, since moving Mendelson to another server is not straightforward.

Once you unzipped and placed the resulting folder on your path of choice (let's call it $tomcat_base_dir), you have the 3 web applications deployed on the same web server. Now you'll need to configure the following files in order to get a working AP:

- *$tomcat_base_dir\bin\certificates.p12*: This is the certificate store Mendelson uses to keep all its partners certificates and your private keypair. You'll just need to import your private keypair into it. All your partner's certificates will be added dynamically into it as you send or receive from them. This store's password is 'test'.
- *$tomcat_base_dir\conf\keystore*: It's the SSL truststore Mendelson uses when sending to partners through SSL. It's already configured with several common root CAs, if you ever face SSL certificate problems when sending to a partner, you'll have to add your partner's root CA certificate here. This store's password is 'test'.
- *$tomcat_base_dir\conf\server.xml*: the central configuration file for your Tomcat server. In this file you can configure what ports the server will use for HTTP and HTTPS. If you use non-standard ports, make sure to open them in your firewall.
- *$tomcat_base_dir\CIPAConf* : in this folder you'll find 3 configuration files for your START AP. All properties have default values and the only ones you need to modify for a correct functioning of your START AP are in configServer.properties, setting *server.keystore.path, server.keystore.password, server.keystore.alias* and *server.keystore.aliaspassword* with the details of your PEPPOL START keypair. To separate START from AS2, you could point this values to *\$tomcat_base_dir\keystores\keystore.jks* , and place your OpenPeppol-issued certificate into it.

- *$tomcat_base_dir\webapps\cipa-dispatcher.war*: open this war and edit the properties file at *\WEB-INF\conf\conf.properties* . Inside you'll find properties to configure your dispatcher:
    - *as2_endpoint_url:* URL of your AS2 endpoint servlet receiving messages. In our case, Mendelson's http receiver.
    - *server_mode*: debug or production, among other things the main difference is on debug mode you can use self-signed SSL certificates.
    - *ssl_truststore , ssl_truststore_password*: only used when your AS2 endpoint uses SSL instead fo plain HTTP. Because most of the times the dispatcher and the AS2 endpoint will be deployed on the same server, you won't need SSL and these two properties will be ignored.
    - *db_driver_name , etc..* : properties configuring Mendelson's HSQLDB where it stores metadata about the partners and message state. You don't need to change these values.
    - *partner_interface_implementation_class , send_interface_implementation_class*: the classes implementing the AS2 interfaces. If you decide to use an AS2 endpoint other than Mendelson, then you'll need to implement IAS2EndpointDBInterface and IAS2EndpointSendInterface from the cipa-dispatcher code, and update these values.
    - *keystore_path, keystore_password*: details about the keystore your AS2 endpoint will use to store your AP keypair and your partners' certificates.
    - *keystore_ap_ca_alias*: alias of the OpenPeppol AP CA certificate inside aforementioned keystore.
    - *keystore_ap_alias*: alias of the OpenPeppol-issued AP certificate representing your endpoint inside aforementioned keystore.
    - *ocsp_responder_url*: URL where the dispatcher will check for revoked certificates, in case *ocsp_validation_activated* is set to true.
    - *temp_folder_path*: the dispatcher automatically creates temp message files every time before sending, this property sets the path for a folder storing them.
    - *mendelson_installation_path*: path where Mendelson's base installation folder is.
    - *mendelson_message_server_host , mendelson_message_server_port*: details of Mendelson's message server.
    - *smp_mode*: the mode in which the dispatcher will determine the receiver's metadata:
        - NO_SMP: the dispatcher will try to retrieve the receiver's metadata from the internal AS2 endpoint's database.
        - DIRECT_SMP: the dispatcher will call the url on *smp_url* to retrieve participants' metadata.
        - PRODUCTION: the dispatcher will use the OpenPeppol infrastructure (DNS discovery + SMP call) to retrieve the participants' metadata.
    - *start_endpoint , as2_endpoint*: the higher the values, the more preference that protocol will have in case you are sending a message to a participant able to communicate in both protocols. If value is set to 0, that protocol will not be enable by the dispatcher.
    - *cache_max_number_entries*: the dispatcher keeps an internal cache with participants' metadata not to have to call the SMP every time. This value sets the maximum number of entries on that cache.

- *cache_expire_entry_after_hours*: number of hours after which the cache will consider an entry expired.

How to use it

1.- SMP installation

After you've finalized your AP configuration, you are ready to make a first test. But to properly test your dispatcher's functionalities you'll first need an accessible SMP with at least one participant registered on it. If you don't have any SMP installed please follow these steps:

- download the SMP web application from JoinUp's Nexus.
- place the war on your web server of choice.
- create your SMP database running this script.
- Inside the war you already placed on your server you'll find *\WEB-INF\classes\config.properties* , where you can configure your SMP keystore and SMP DB parameters.
- now the SMP application is ready to run, start the server where the SMP is deployed.
- download the released SMP test project for SoapUI. Import this project into the freely available SoapUI tool.
- once in SoapUI, double click on the SMP node marked in blue, and under the Service Endpoints tab modify the url to point to your just started SMP.
- now we are going to register a participant into your SMP. First step is put a ServiceGroup, go to 'Non-Core SMP Services' node and double click the test 'Put ServiceGroup'. Change the participant's data at your convenience, making sure the url parameters (participant scheme and participant id in this case) match the data you insert in the XML. Run the test.
- after having our ServiceGroup, we are now going to create the service metadata. Double click on 'Put SignedServiceMetadata' test. If you changed the values for the service group, you'll have to make the same changes here. The xml field 'EndpointReference' is the url where messages will be sent for this participant. The attribute *transportProfile* in the *Endpoint* tag is the protocol this participant will be registered with. Possible values as specified in the "Peppol Policy for Identifiers" document are *busdox-transport-start* for START protocol, *busdox-transport-as2-ver1p0* for AS2 protocol. Run the test.
- the participant has been correctly registered on your SMP, now you can test the dispatcher.

2.- Testing the dispatcher

Now we are going to send our first AS2 message through the dispatcher:

- make sure that config.properties file inside cipa-dispatcher.war has the values for *smp-mode* set to direct_smp and *smp_url* pointing to your SMP.

- once the server where you placed your dispatcher is running, it'll be listening for messages to send on /cipa-dispatcher/send.
- download our released AS2 SoapUI project and import it into SoapUI.
- double click the AS2 node marked with a blue icon and under Service Endpoints tab change the url if needed.
- double click on the 'Send AS2 Message' test. Here you'll have to make modifications accordingly to the participant you created on your SMP, so that Recevier.Identifier , DocumentID.InstanceIdentifier and ProcessID.InstanceIdentifier have the right values you set when you registered its metadata.
- run the test. If everything has been correctly set up, it will call the dispatcher, that will extract the receiver's values from the SBDH header and call your SMP to retrieve the participant's metadata. Once the dispatcher knows the receiver's certificate and url where to send, it updates Mendelson with those datas and forwards the message to it. The message should arrive to your AS2 receiver endpoint few seconds afterwards.

When sending to the dispatcher, you can get the following error messages due to bad SBDH documents or misconfiguration:

- *An error occured while treating the SBDH request*: your SBDH document is malformed. You'll receive information about what's missing in the error response.
- *The necessary field 'X' could not be found in the SBDH header*: one of RecevierIdentifier, SenderIdentifier, DocumentIdentifier or ProcessIdentifier weren't included on your SBDH document, or they weren't at the right xml location.
- *There's no activated protocol in this Access Point able to communicate with the receiver*: the protocols you want to activate on your dispatcher are specified in conf.properties inside cipa-dispatcher.war. If the receiver of your message doesn't use any of your activated protocols, there's no way your endpoint can send to it.

If you want to test message reception on your new AP, you'll have to register the followings urls on an SMP:

  - host:port/cipa-dispatcher/AS2Receiver for AS2 protocol
  - host:port/cipa-start-server/accessPointService for START protocol.

Message reception is automatic and there's no need for any action from your side. If a message signed with a valid PEPPOL-issued AP certificate gets to your AP you'll successfully receive the message, and it'll be stored in your server (by default inside */bin/messages/* for AS2 messages and inside */bin/server-workdir/* for START messages).
More information about Mendelson AS2 endpoint can be found at http://community.mendelson-e-c.com/forum/as2.
If you want to learn more about how AS2 protocol works, please refer to http://en.wikipedia.org/wiki/AS2 and http://www.ietf.org/rfc/rfc4130.txt .