django-pyas2

Release 1.2.3

Abhishek Ram

Feb 25, 2023

CONTENTS

Release v1.2.3. (*Changelog*)

`django-pyas2` is an AS2 server/client written in python and built on the Django framework. The application supports AS2 version 1.2 as defined in the RFC 4130. Our goal is to provide a native python library for implementing the AS2 protocol. It supports Python 3.6+.

The application includes a server for receiving files from partners, a front-end web interface for configuration and monitoring, a set of `django-admin` commands that serves as a client for sending messages, asynchronous MDNs and a daemon process that monitors directories and sends files to partners when they are placed in the partner's watched directory.

CHAPTER

ONE

FEATURES

- Technical
  - Asynchronous and Synchronous MDN
  - Partner and Organization management
  - Digital signatures
  - Message encryption
  - Secure transport (SSL)
  - Support for SSL client authentication
  - System task to auto clear old log entries
  - Data compression (AS2 1.1)
  - Multinational support: Uses Django's internationalization feature
- Integration
  - Easy integration to existing systems, using a partner based file system interface
  - Message post processing (scripting on receipt)
- Monitoring
  - Web interface for transaction monitoring
  - Email event notification
- The following encryption algorithms are supported:
  - Triple DES
  - RC2-128
  - RC4-128
  - AES-128
  - AES-192
  - AES-256
- The following hash algorithms are supported:
  - SHA-1
  - SHA-224
  - SHA-256

- SHA-384
- SHA-512

CHAPTER

TWO

DEPENDENCIES

- Python 3.6+
- Django (1.9+)
- requests
- pyas2lib

CHAPTER

THREE

GUIDE

3.1 Installation

Install using `pip`...

```
$ pip install django-pyas2
```

Create a new `django` project

```
$ django-admin startproject django_pyas2 .
```

Add `pyas2` to your `INSTALLED_APPS` setting.

```
INSTALLED_APPS = (
    ...
    'pyas2',
)
```

Include the pyAS2 URL configuration in your project's `urls.py`.

```
from django.urls import include
urlpatterns = [
    path('pyas2/', include('pyas2.urls')),
    ...
]
```

Run the following commands to complete the installation and start the server.

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, pyas2, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
```

(continues on next page)

(continued from previous page)

```
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying pyas2.0001_initial... OK
  Applying sessions.0001_initial... OK

$ python manage.py createsuperuser
Username (leave blank to use 'abhishekram'): admin
Email address: admin@domain.com
Password:
Password (again):
Superuser created successfully.

$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 01, 2019 - 07:33:27
Django version 2.2, using settings 'django_pyas2.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

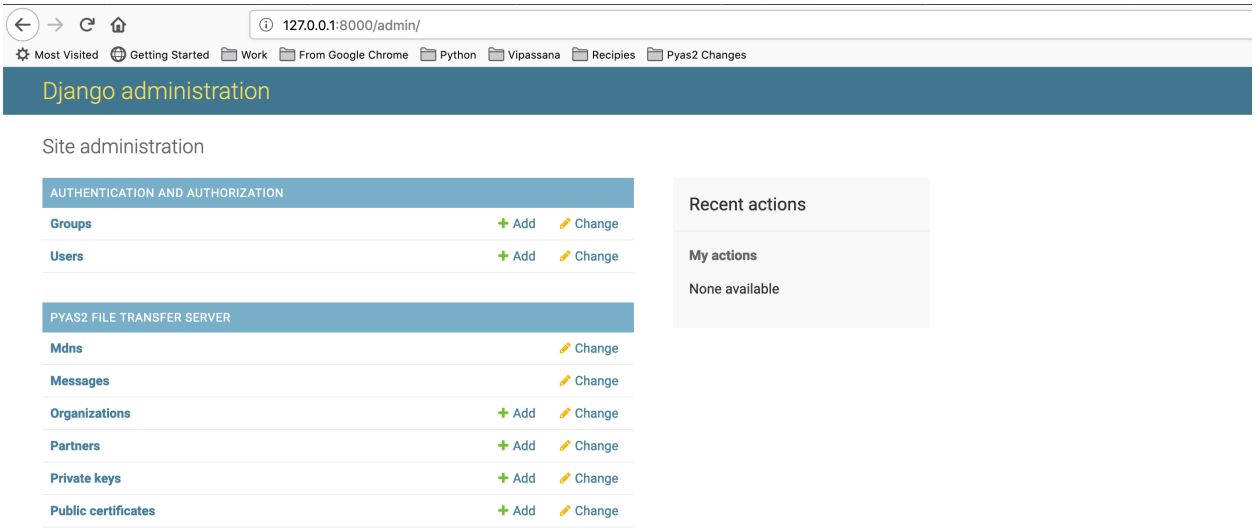
The `django-pyas2` server is now up and running, the web UI for configuration and monitoring can be accessed at `http://localhost:8000/admin/pyas2/` and the endpoint for receiving AS2 messages from your partners will be at `http://localhost:8080/pyas2/as2receive`

3.2 Quickstart

This guide will walk you through the basics of setting up an AS2 server and transferring files using the AS2 protocol. Let's get started by sending a signed and encrypted file from one pyAS2 server `P1` to another pyAS2 server `P2`. Do note that these two are separate installations of pyAS2.

3.2.1 Installing the Servers

Create a Django project called `P1` and follow the *installation guide* and run `python manage.py runserver` to start `P1` at http://127.0.0.1:8000/admin/



Create one more Django project called `P2` and follow the same installations instructions, and run `python manage.py runserver 127.0.0.1:8001` to start `P2` at http://127.0.0.1:8001/admin/

3.2.2 Creating the certificates

We need to generate a Public and Private key pair each for the two servers. `P1` uses its private key to sign the message which is verified by `P2` using `P1's` public key. `P1` uses the `P2's` public key to encrypt the message which is decrypted by `P2` using its private key.

To generate the public and private key pair use the below commands

```
$ openssl req -x509 -newkey rsa:2048 -sha256 -keyout P1_private.pem -out P1_public.
↪pem -days 365
Generating a 2048 bit RSA private key
.....+++
.....
↪..........+++
writing new private key to 'P1_private.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:Bangalore
Organization Name (eg, company) [Internet Widgits Pty Ltd]:P1
Organizational Unit Name (eg, section) []:AS2
Common Name (e.g. server FQDN or YOUR name) []:p1as2
Email Address []:
$ cat P1_public.pem >> P1_private.pem
```

(continues on next page)

(continued from previous page)

```
$ openssl req -x509 -newkey rsa:2048 -sha256 -keyout P2_private.pem -out P2_public.
↪pem -days 365
Generating a 2048 bit RSA private key
..............................+++
.....................................................................................
↪.....................+++
writing new private key to 'P2_private.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:Bangalore
Organization Name (eg, company) [Internet Widgits Pty Ltd]:P2
Organizational Unit Name (eg, section) []:AS2
Common Name (e.g. server FQDN or YOUR name) []:p2as2
Email Address []:
$ cat P2_public.pem >> P2_private.pem
```

3.2.3 Configure P1

`P1` needs to be configured before it can start sending files, open the web UI and follow these instructions:

- Navigate to `Private Keys->Add private key`.
- Choose the file `P1_private.pem` in the *key file* field, enter the passphrase and save the Private Certificate.
- Next navigate to `Public Certificates->Add public certificate`.
- Choose the file `P2_public.pem` in the *certificate file* field and save the Public Certificate.
- Now navigate to `Organization->Add organization`.
- Set Name to `P1`, As2 Name to `p1as2` and set the Signature and Encryption keys to `P1_private.pem` and save the Organization.
- Next navigate to `Partner->Add partner`.
- Set Name to `P2`, As2 Name to `p2as2` and Target url to `http://127.0.0.1:8001/pyas2/as2receive`
- Under security settings set Encrypt Message to `3DES`, Sign Message to `SHA-256`, Signature and Encryption keys to `P2_public.pem`.
- Under MDN settings set MDN mode to `Synchronous` and Request Signed MDN to `SHA-256`.
- Save the partner to complete the configuration.

3.2.4 Configure P2

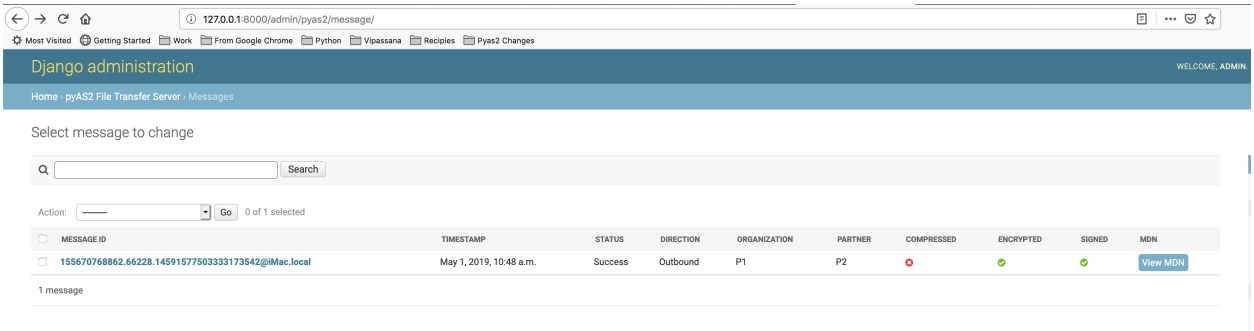
P2 needs to be configured before it can start receiving files, open the web UI and follow these instructions:

- Navigate to Private Certificates->Add private key.
- Choose the file P2_private.pem in the *key file* field, enter the passphrase and save the Private Certificate.
- Next navigate to ublic Certificates->Add public certifcate.
- Choose the file P1_public.pem in the *certificate file* field and save the Public Certificate.
- Now navigate to Organization->Add organization.
- Set Name to P2, As2 Name to p2as2 and set the Signature and Encryption keys to P2_private.pem and save the Organization.
- Next navigate to Partner->Add partner.
- Set Name to P1, As2 Name to p1as2 and Target url to http://127.0.0.1:8000/pyas2/as2receive
- Under security settings set Encrypt Message to 3DES, Sign Message to SHA-256, Signature and Encryption keys to P1_public.pem.
- Under MDN settings set MDN mode to Synchronous and Request Signed MDN to SHA-256.
- Save the partner to complete the configuration.

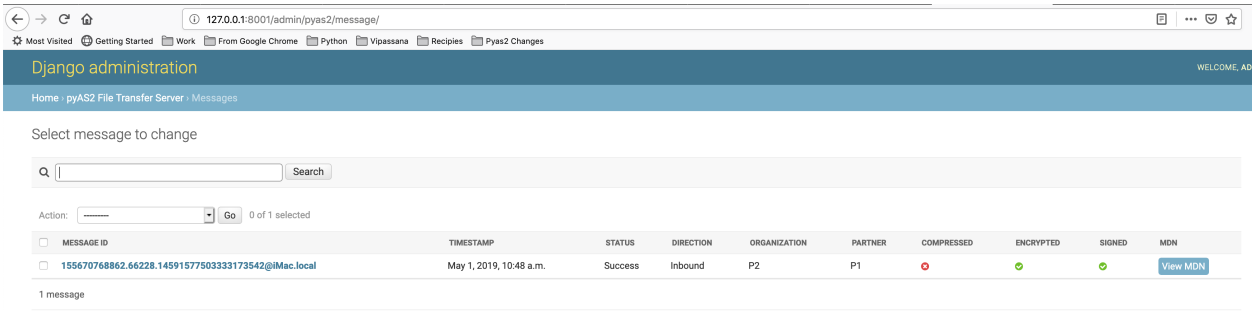
3.2.5 Send a File

We are now read to send a file from P1 to P2, to do so follow these steps:

- Open the P1 web UI and navigate to *Partners*'.
- Select the partner *P2* and action *Send a message to selected partner* and click Go.
- Select the Organization as P1 and Partner as P2.
- Now select the file to send and click Send Message.
- The status of the file transfer can be viewed by navigating to Messages.
- Once file transfer is completed you will a green tick in the status column.



- We will also see a similar entry in the web UI of P2.



- We can see basic information on this screen such as Partner, Organization, Message ID and MDN.
- We can also view the MDN and Payload by clicking on the respective links.

3.2.6 Conclusion

We have successfully demonstrated the core functionality of `django-pyas2` i.e. sending files from one system to another using the AS2 protocol. For a more detailed overview of all its functionality do go through the *detailed docs*.

3.3 Detailed Guide

We have seen how to send a file to the partner with the basic settings. Now lets go through each of the components of `django-pyas2` in greater detail. In this section we will cover topics related to configuration of partners, organizations and certificates; sending messages and MDNs; monitoring messages and MDNs; and usage of the admin commands.

3.3.1 Organizations

Organizations in `django-pyas2` mean the host of the AS2 server, i.e. it is the sender when sending messages and the receiver when receiving the messages. Organizations can be managed from the Django Admin. The admin lists the existing organizations and also you gives the option to create new ones. Each organization is characterized by the following fields:

Field Name	Description	Mandatory
`Organization Name`	The descriptive name of the organization.	Yes
`As2 Identifier`	The as2 identifies for this organization, must be a unique value as it identifies the as2 host.	Yes
`Email Address`	The email address for the organization.	No
`Encryption Key`	The `Private Key` used for decrypting incoming messages from trading partners.	No
`Signature Key`	The `Private Key` used to sign outgoing messages to trading partners	No
`Confirmation Message`	Use this field to customize the confirmation message sent in MDNs to partners.	No

3.3.2 Partners

Partners in `django-pyas2` mean all your trading partners with whom you will exchanges messages, i.e. they are the receivers when you send messages and the senders when you receive messages. Partners can be managed from the Django Admin. The admin lists the existing partners and also you gives the option to search them and create new ones. Each partner is characterized by the following fields:

General Settings

Field Name	Description	Mandatory
`Partner Name`	The descriptive name of the partner.	Yes
`As2 Identifier`	The as2 identifies for this partner as communicated by the partner.	Yes
`Email Address`	The email address for the partner.	No
`Target Url`	The HTTP/S endpoint of the partner to which files need to be posted.	Yes
`Subject`	The MIME subject header to be sent along with the file.	Yes
`Content Type`	The content type of the message being transmitted, can be XML, X12 or EDIFACT.	Yes
`Confirmation Message`	Use this field to customize the confirmation message sent in MDNs to partners.	No

HTTP Authentication

Use these settings if basic authentication has been enabled for the partners AS2 server.

Field Name	Description	Mandatory
`Enable Authentication`	Check this option to enable basic AUTH.	No
`Http auth user`	User name to access the partners server.	No
`Http auth pass`	Password to access the partners server.	No

Security Settings

Field Name	Description	Mandatory
`Compress Message`	Check this option to enable AS2 message compression.	No
`Encrypt Message`	Select the algorithm to be used for encrypting messages, defaults to None.	No
`Encryption Key`	Select the `Public Key` used for encrypting the outbound messages to this partner.	No
`Sign Message`	Select the hash algorithm to be used for signing messages, defaults to None. incoming messages from trading partners.	No
`Signature key`	The `Public Key` used to verify inbound signed messages and MDNs from this partner	No

MDN Settings

Field Name	Description	Mandatory
`Request MDN`	Check this option to request MDN for outbound messages to this partner.	Yes
`Mdn mode`	Select the MDN mode, defaults to Synchronous	No
`Request Signed MDN`	Select the algorithm to be used in case signed MDN is to be returned.	No

Advanced Settings

Field Name	Description	Mandatory
`Keep Original Filename`	Use Original File name to to store file on receipt, use this option only if you are sure partner sends unique names.	No
`Command on Message Send`	OS Command executed after successful message send, replacements are `$filename`, `$sender`, `$receiver`, `$messageid` and any message header such as `$Subject`	No
`Command on Message Receipt`	OS Command executed after successful message receipt, replacements are `$filename`, `$fullfilename`, `$sender`, `$receiver`, `$messageid` and any message header such as `$Subject`.	No

3.3.3 Keys & Certificates

The AS2 protocol strongly encourages the use of RSA certificates to sign and encrypt messages for enhanced security. A signed and encrypted message received from your partner ensures message repudiation and integrity. The RSA certificate consists of a public key and a private key which are together used for encrypting, decrypting, signing and verifying messages.

Generating Certificates

When you set up a new AS2 server you will need to generate a Public/Private key pair. The private key will be added to your server and the public key needs to be shared with your trading partners.

One of the ways of generating a certificate is by using the `openssl` command line utility, the following command needs to be used:

```
$ openssl req -x509 -newkey rsa:2048  -sha256 -keyout private.pem -out public.pem -
↪days 365
Generating a 2048 bit RSA private key
.....+++
.....
↪..........+++
writing new private key to 'private.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
```

(continues on next page)

(continued from previous page)

```
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:Bangalore
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Name
Organizational Unit Name (eg, section) []:AS2
Common Name (e.g. server FQDN or YOUR name) []:as2id
Email Address []:
$ cat public.pem >> private.pem
```

The above commands will generate a PEM encoded private key called `private.pem` and a PEM encoded public key called `public.pem`.

Private Keys

`Private Keys` are used for signing outbound messages to your partners and decrypting incoming messages from your partners. We can manage them in `django-pyas2` from the Django Admin. The admin lists all your private keys and lets you add new ones. Each `Private Key` is characterized by the following fields:

Field Name	Description	Mandatory
`Key File`	Select the **PEM** or **DER** encoded[1] private key file[2].	Yes
`Private Key Password`	The pass phrase entered at the time of the certificate generation.	Yes

Public Certificates

`Public Certificates` are used for verifying signatures of inbound messages and encrypting outbound messages to your partners. The public key file will be shared by your partner. We can manage them in `django-pyas2` from the Django Admin. The admin screen lists all your public certificates and lets you add new ones. Each `Public Certificate` is characterized by the following fields:

Field Name	Description	Mandatory
`Certificate File`	Select the **PEM** or **DER** encoded[1] public key file.	Yes
`Certificate CA Store`	In case the certificate has been signed by an unknown CA then select the CA certificate here.	No
`Verify Certificate`	Uncheck this option to disable certificate verification at the time of signature verification.	No

3.3.4 Configuration

The global settings for `pyAS2` are kept in a single configuration dictionary named `PYAS2` in your project's settings.py module. Below is a sample configuration:

[1] `django-pyas2` supports only PEM/DER encoded certificates.
[2] The private key file must contain **both the private and public** parts of the RSA certificate.

```
PYAS2 = {
    'DATA_DIR' : '/path_to_datadir/data',
    'MAX_RETRIES': 5,
    'MDN_URL' : 'https://192.168.1.115:8888/pyas2/as2receive',
    'ASYNC_MDN_WAIT' : 30,
    'MAX_ARCH_DAYS' : 30,
}
```

The available settings along with their usage is described below:

Settings Name	Default Value	Usage
DATA_DIR	MEDIA_ROOT or BASE_DIR	Full path to the base directory for storing messages
MAX_RETRIES	10	Maximum number of retries for failed outgoing messages
MDN_URL	None	Return URL for receiving asynchronous MDNs from partners.
ASYNC_MDN_WAIT	30	Number of minutes to wait for asynchronous MDNs after which message will be marked as failed.
MAX_ARCH_DAYS	30	Number of days files and messages are kept in storage.

The Data Directory

The `Data Directory` is a file system directory that stores sent and received files. The location of this directory is set to either the `MEDIA_ROOT` or the project base folder by default. We can also change this directory by updating the `DATA_DIR` setting. The structure of the directory is below:

```
{DATA DIRECTORY}
└── messages
    ├── __store
    │   ├── mdn
    │   │   ├── received
    │   │   └── sent
    │   │       ├── 20150908
    │   │       │   ├── 20150908115337.7244.44635@Abhisheks-MacBook-Air.local.mdn
    │   │       │   └── 20150908121942.7244.71894@Abhisheks-MacBook-Air.local.mdn
    │   │       └── 20150913
    │   │           ├── 20150913071324.20065.47671@Abhisheks-MacBook-Air.local.mdn
    │   │           └── 20150913083125.20403.32480@Abhisheks-MacBook-Air.local.mdn
    │   └── payload
    │       ├── received
    │       │   ├── 20150908
    │       │   │   ├── 20150908115458.7255.98107@Abhisheks-MacBook-Air.local
    │       │   │   └── 20150908121933.7343.83150@Abhisheks-MacBook-Air.local
    │       │   └── 20150913
    │       │       ├── 20150913071323.20074.48016@Abhisheks-MacBook-Air.local
    │       │       └── 20150913083125.20475.14667@Abhisheks-MacBook-Air.local
    │       └── sent
    ├── p1as2
    │   └── outbox
    │       └── p2as2
    └── p2as2
        └── inbox
            └── p1as2
                ├── 20150908115458.7255.98107@Abhisheks-MacBook-Air.local.msg
                └── 20150913083125.20475.14667@Abhisheks-MacBook-Air.local.msg
```

inbox

The inbox directory stores files received from your partners. The path of this directory is `{DATA DIRECTORY}/messages/{ORG AS2 ID}/inbox/{PARTNER AS2 ID}`. We need to take this location into account when integrating `django-pyas2` with other applications.

outbox

The outbox directory works in conjunction with the `sendas2bulk` process. The bulk process looks in all of the outbox directories and will trigger a transfer for each file found. The path of this directory is `{DATA DIRECTORY}/messages/{PARTNER AS2 ID}/outbox/{ORG AS2 ID}`.

__store

The __store directory contains the payloads and MDNs. The payload and MDN files are stored in the sent and received sub-directories respectively, and are further seperated by additional sub-directories for each day, named as YYYYMMDD.

3.3.5 Send & Receive Messages

We have so far covered all the topics related to configuration of the `pyAS2` server. Now we will see how to use these configurations to send messages to your trading partners using the AS2 protocol. We can send files using any of the following techniques:

Send Messages From the Django Admin

The simplest method for sending messages to your trading partner is by using the Django Admin. This method is generally used for testing the AS2 connection with your trading partner. The steps are as follows:

- Navigate to `pyAS2 File Transfer Server->Partners`.
- Check the partner you want to send the message to and select action `Send Message to Partner`.
- Select the Organization and choose the file to be transmitted.
- Click on `Send Message` to initiate the file transfer and monitor the transfers at `pyAS2 File Transfer Server->Messages`.

Send Messages From the Command-Line

The next method for sending messages involves the `django-pyas2` admin command `sendas2message`. The command is invoked from the shell prompt and can be used by other applications to invoke an AS2 file transfer. The command usage is as follows:

```
$ python manage.py sendas2message --help
Usage: python manage.py sendas2message [options] <organization_as2name partner_
↪as2name path_to_payload>

Send an as2 message to your trading partner

Options:
  --delete              Delete source file after processing
  -h, --help            show this help message and exit
```

The mandatory arguments to be passed to the command include `organization_as2name` i.e. the AS2 Identifier of this organization, `partner_as2name` i.e. the AS2 Identifier of your trading partner and `path_to_payload` the full path to the file to be transmitted. The command also lets you set the `--delete` option to delete the file once it is begins the transfer. A sample usage of the command:

```
$ python manage.py sendas2message p1as2 p2as2 /path_to_payload/payload.txt
```

Receive Messages

In order to receive files from your trading partners they need to post the AS2 message to the URL `http://{hostname}:{port}/pyas2/as2receive`. The configuration of the *Organization*, *Partner* and *Certificates* need to be completed for successfully receiving messages from your trading partner. Once the message has been received it will be placed in the organizations inbox folder.

3.3.6 Send & Receive MDNs

Message Disposition Notifications or MDNs are return receipts used to notify the sender of a message of any of the several conditions that may occur after successful delivery. In the context of the AS2 protocol, the MDN is used to notify if the message was successfully processed by the receiver's system or not and in case of failures the reason for the failure is sent with the MDN.

MDNs can be transmitted either in a synchronous manner or in an asynchronous manner. The synchronous transmission uses the same HTTP session as that of the AS2 message and the MDN is returned as an HTTP response message. The asynchronous transmission uses a new HTTP session to send the MDN to the original AS2 message sender.

Send MDNs

The choice of whether to send an MDN and its transfer mode is with the sender of the AS2 message. The sender lets us know what to do through an AS2 header field. In case the partner requests a synchronous MDN no action is needed as `django-pyas2` takes care of this internally, however in the case of an asynchronous MDN the admin command `manageas2server --async-mdns` needs to be run to send the MDN to the trading partner.

The command `{PYTHONPATH}/python {DJANGOPROJECTPATH}/manage.py manageas2server --async-mdns` should be scheduled every 10 minutes so that `django-pyas2` sends any pending asynchronous MDN requests received from your trading partners.

Receive MDNs

The choice of whether or not to receive MDN and its transfer mode is with us. The MDN Settings for the partner should be used to specify your preference. In case of synchronous mode `django-pyas2` processes the received MDN without any action from you.

In the case of asynchronous mode we do need to take care of a couple of details to enable the receipt of the MDNs. The *global setting* `MDNURL` should be set to the URL `http://{hostname}:{port}/pyas2/as2receive` so that the trading partner knows where to send the MDN. The other setting of note here is the `ASYNCMDNWAIT` that decides how long `django-pyas2` waits for an MDN before setting the message as failed so that it can be retried. The admin command `manageas2server --async-mdns` makes this check for all pending messages so it must be scheduled to run regularly.

3.3.7 Admin Commands

`django-pyas2` provides a set of Django `manage.py` admin commands that perform various functions. We have already seen the usage of some of these commands in the previous sections. Let us now go through the list of available commands:

sendas2message

The `sendas2message` command triggers a file transfer, it takes the mandatory arguments organization id, partner id and the full path to the file to be transferred. The command can be used by other applications to integrate with `django-pyas2`.

sendas2bulk

The `sendas2bulk` command looks in the outbox folder for each partner setup on the as2 server. It then triggers a transfer for each file found in the outbox.

manageas2server

The `manageas2server` command performs various management operation on the AS2 server. The following options are available which can either be used together or alone:

- `--async-mdns`: This operation performs two functions; it sends asynchronous MDNs for messages received from your partners and also checks if we have received asynchronous MDNs for sent messages so that the message status can be updated appropriately.
- `--retry`: This operation checks for any messages that have been set for retries and then re-triggers the transfer for these messages.
- `--clean`: This operation deletes all messages objects and related files older that the `MAX_ARCH_DAYS` setting.

3.3.8 Docker

`django-pyas2` can easily be run as a docker container. Following instruction can be used to configure a Dockerfile for the application.

The assumption is that a directory containtain the django-project exists already, as described in the installation section. Create a Dockerfile in the project path and the directory should look as follows:

```
{PROJECT DIRECTORY}
└──django_pyas2
    ├── django_pyas2
    │   ├── db.sqlite3
    │   ├── manage.py
    │   └── django_pyas2
    │       ├── settings.py
    │       ├── urls.py
    │       └── wsgi.py
    └── Dockerfile
```

Populate the Dockerfile with following content:

```
FROM python:3.7-alpine3.9

# Update the index of available packages
RUN apk update

# Install packages required for Python cryptography
RUN apk add --no-cache openssl-dev gcc libffi-dev musl-dev

# Install django-pyas2 with pip
RUN pip install django-pyas2

# Copy the files from the project directory to the container
WORKDIR /
COPY django_pyas2 django_pyas2
CMD ["/usr/local/bin/python", "/django_pyas2/manage.py", "runserver", "0.0.0.0:8000"]

# AS2 Server
EXPOSE 8000
```

Then build and run the container from the command line as follows:

```
$ docker build -t docker_pyas2 . && docker run -p 8000:8000 docker_pyas2
```

In case the files on the host file system should be used, connect the directory to the host by running to docker run command with the -v option:

```
$ docker build -t docker_pyas2 . && docker run -p 8000:8000 -v $PWD/django_pyas2:/
↪django_pyas2 docker_pyas2
```

3.3.9 Extending django-pyas2

A use case for extending django-pyas2 may be to have additional connectors, from which files are received, such as a message queue, or to run a directory monitor as a daemon to send messages as soon as a message has been written to an outbound directory (see directory structure), or to add additional functionalities, like a custom website to the root of the url etc.

One way to extend `django-pyas2` is to use the django startapp command, that will create the directory structure needed for an app. In this example we call the app "extend_pyas2".

Please consult the extensive django documentation to learn more about these command. Below simply a description for your convenience to get started:

In the django_pyas2 project directory invoke the script as follows:

```
$ python manage.py startapp extend_pyas2
```

This has now created a new directory containing files that may be used for apps:

```
{PROJECT DIRECTORY}
└──django_pyas2
    ├── django_pyas2
    │   ├── db.sqlite3
    │   ├── manage.py
    │   └── django_pyas2
    │       ├── settings.py
    │       ├── urls.py
    │       └── wsgi.py
    └── extend_pyas2
        ├── apps.py
        ├── migrations
        ├── models.py
        ├── tests.py
        └── views.py
```

In our example, we will add a new admin command that should monitor directories and trigger the sending of files to partners when they are written. For that purpose, we need to create some subfolders "management/commands" and a python file with the management command:

```
│       └── wsgi.py
└── extend_pyas2
    ├── apps.py
    ├── migrations
    ├── models.py
    ├── tests.py
    ├── views.py
    └── management
        └── commands
            └── filewatcher.py
```

Add `extend_pyas2` to your `INSTALLED_APPS` settings, after `pyas2`.

```
INSTALLED_APPS = (
    ...
    'pyas2',
    'extend_pyas2',
)
```

An example content for the filewatcher.py may be as follows and can be run with Django's manage command:

```
$ python manage.py filewatcher
```

```
from django.core.management.base import BaseCommand, CommandError
from django.core.management import call_command
from django.utils.translation import ugettext as _
from pyas2.models import Organization
from pyas2.models import Partner
from pyas2 import settings
from watchdog.observers import Observer
from watchdog.observers.polling import PollingObserverVFS
from watchdog.events import PatternMatchingEventHandler
import time
```

(continues on next page)

(continued from previous page)

```
import atexit
import socket
import os
import sys
import logging

logger = logging.getLogger('django')

DAEMONPORT = 16388


class FileWatchHandle(PatternMatchingEventHandler):
    """
    FileWatchHandler that ignores directories. No Patterns defined by default. Any␣
↪file in the
    directory will be sent.
    """
    def __init__(self, tasks, dir_watch):
        super(FileWatchHandle, self).__init__(ignore_directories=True)
        self.tasks = tasks
        self.dir_watch = dir_watch

    def handle_event(self, event):
        self.tasks.add(
            (self.dir_watch['organization'], self.dir_watch['partner'], event.src_
↪path))
        logger.info(u' "%(file)s" created. Adding to Task Queue.', {'file': event.src_
↪path})

    def on_modified(self, event):
        self.handle_event(event)

    def on_created(self, event):
        self.handle_event(event)


class WatchdogObserversManager:
    """
    Creates and manages a list of watchdog observers as daemons. All daemons will␣
↪have the same
    settings. By default, subdirectories are not searched.
    :param: force_vfs : if the underlying filesystem is a network share, OS events␣
↪cannot be
                        used reliably. Polling to be done, which is expensive.
    """
    def __init__(self, is_daemon=True, force_vfs=False):
        self.observers = []
        self.is_daemon = is_daemon
        self.force_vfs = force_vfs

    def add_observer(self, tasks, dir_watch):
        if self.force_vfs:
            new_observer = PollingObserverVFS(stat=os.stat, listdir=os.listdir)
        else:
            new_observer = Observer()
        new_observer.daemon = self.is_daemon
        new_observer.schedule(FileWatchHandle(tasks, dir_watch),
```

(continues on next page)

(continued from previous page)

```
                                dir_watch['path'], recursive=False)
        new_observer.start()
        self.observers.append(new_observer)

    def stop_all(self):
        for observer in self.observers:
            observer.stop()

    def join_all(self):
        for observer in self.observers:
            observer.join()


class Command(BaseCommand):
    help = _(u'Daemon process that watches the outbox of all as2 partners and '
             u'triggers sendmessage when files become available')

    def handle(self, *args, **options):
        logger.info(_(u'Starting PYAS2 send Watchdog daemon.'))
        engine_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            engine_socket.bind(('127.0.0.1', DAEMONPORT))
        except socket.error:
            engine_socket.close()
            raise CommandError(_(u'An instance of the send daemon is already running
↪'))
        else:
            atexit.register(engine_socket.close)

        tasks = set()
        dir_watch_data = []

        for partner in Partner.objects.all():
            for org in Organization.objects.all():
                outboxDir  = os.path.join(settings.DATA_DIR,
                                          'messages',
                                          partner.as2_name,
                                          'outbox',
                                          org.as2_name)
                if os.path.isdir(outboxDir):
                    dir_watch_data.append({})
                    dir_watch_data[-1]['path'] = outboxDir
                    dir_watch_data[-1]['organization'] = org.as2_name
                    dir_watch_data[-1]['partner'] = partner.as2_name

        if not dir_watch_data:
            logger.error(_(u'No partners have been configured!'))
            sys.exit(0)

        logger.info(_(u'Process existing files in the directory.'))
        for dir_watch in dir_watch_data:
            files = [f for f in os.listdir(dir_watch['path']) if
                     os.path.isfile(os.path.join(dir_watch['path'], f))]
            for file in files:
                logger.info(u'Send as2 message "%(file)s" from "%(org)s" to "
↪%(partner)s".',
                            {'file': file,
```

(continues on next page)

(continued from previous page)

```
                            'org': dir_watch['organization'],
                            'partner': dir_watch['partner']})

                call_command('sendas2message', dir_watch['organization'], dir_watch[
↪'partner'],
                            os.path.join(dir_watch['path'], file), delete=True)

        """Add WatchDog Thread Here"""
        logger.info(_(u'PYAS2 send Watchdog daemon started.'))
        active_receiving = False
        watchdog_file_observers = WatchdogObserversManager(is_daemon=True, force_
↪vfs=True)
        for dir_watch in dir_watch_data:
            watchdog_file_observers.add_observer(tasks, dir_watch)
        try:
            logger.info(_(u'Watchdog awaiting tasks...'))
            while True:
                if tasks:
                    if not active_receiving:
                        # first request (after tasks have been fired, or startup of␣
↪dirmonitor)
                        active_receiving = True
                    else:  # active receiving events
                        for task in tasks:
                            logger.info(
                                u'Send as2 message "%(file)s" from "%(org)s" to "
↪%(partner)s".',
                                {'file': task[2],
                                 'org': task[0],
                                 'partner': task[1]})

                            call_command('sendas2message', task[0], task[1], task[2],
                                         delete=True)
                        tasks.clear()
                        active_receiving = False
                time.sleep(2)

        except (Exception, KeyboardInterrupt) as msg:
            logger.info(u'Error in running task: "%(msg)s".', {'msg': msg})
            logger.info(u'Stopping all running Watchdog threads...')
            watchdog_file_observers.stop_all()
            logger.info(u'All Watchdog threads stopped.')

        logger.info(u'Waiting for all Watchdog threads to finish...')
        watchdog_file_observers.join_all()
        logger.info(u'All Watchdog threads finished. Exiting...')
        sys.exit(0)
```

3.4 Release History

3.4.1 1.2.3 - 2023-02-25

- Bump version of pyas2lib to 1.4.3
- Update variables in run_post_receive to fit the meaning (#82 by @timfanda35)

- Fix link to AUTHORS since now it's a Markdown file (#85 by @adiroiban)
- Update the lengths of the payload fields to allow longer file names (#87 by @pouldenton)
- Update documentation to use django-admin instead of django-admin.py (#89 by @bkc)

3.4.2 1.2.2 - 2022-02-06

- Bump version of pyas2lib to 1.4.0 (PR #70)
- Use github actions for running test pipeline instead of travis
- Add support for python 3.10 and upgrade pytest* packages
- Deprecate support for python 3.6
- Replace deprecated ugettext with gettext_lazy (PR #68 by @liquidxinc)

3.4.3 1.2.1 - 2021-05-08

- Bump version of pyas2lib to 1.3.3
- Use orig_message_id as Message ID for MDN if no message_id was provided
- Retry when no ASYNC MDN is received, before finally failing after retries
- Bump version of django to 2.2.18

3.4.4 1.2.0 - 2020-04-12

- Bump version of pyas2lib to 1.3.1
- Improve the test coverage for the repo
- Use django storage framework when dealing with the file system
- Handle cases where we get a 200 response without an MDN when sending messages
- Set login required for the download and send message endpoints

3.4.5 1.1.1 - 2019-06-25

- Bump version of pyas2lib to 1.2.2
- Add more logging for better debugging
- Removing X-Frame-Options header from AS2 response object

3.4.6 1.1.0 - 2019-06-13

- Use original filename when saving to store and allow search by filename.
- Bump version of pyas2lib to 1.2.0 to fix issue #5
- Minimum version of django is now 2.1.9 which fixes issue #8
- Extract and save certificate information on upload.

3.4.7 1.0.2 - 2019-05-16

- Add command *sendas2bulk* for sending messages in the outbox folders.
- Add command *manageas2server* for cleanup, async mdns and retries.

3.4.8 1.0.1 - 2019-05-02

- Use current date as sub-folder in message store
- Use password widget for *key_pass* field of PrivateKey
- Better rendering of headers and payload in messages
- Include templates in the distribution

3.4.9 1.0.0 - 2018-05-01

- Initial release.