

---

# **ODST Documentation**

***Release***

**Bryson Tyrrell**

**Jan 15, 2018**



<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>The Open Distribution Server</b>	<b>5</b>
<b>3</b>	<b>Completed Features and Notes</b>	<b>17</b>
<b>4</b>	<b>Contributing to ODST</b>	<b>19</b>
	<b>HTTP Routing Table</b>	<b>21</b>



The ODST project aims to deliver an open and automated file distribution solution for IT administrators. The core application of this solution is the Open Distribution Server (ODS).

**Warning:** This project is currently in an Alpha state and is being actively developed and tested. See **Contributing to ODST** below to learn more about how you can help!



# CHAPTER 1

---

## Features

---

- Secure and Automatic Bi-Directional File Syncing
- Web Interface
- Admin API for Integrations
- File and Server Stage Tagging
- Webhook and Email Notification Options
- Support for Jamf Pro
- And More...

---

**Note:** If you use a management solution other than Jamf Pro and would like to see integration features between ODST and your solution, please open a GitHub issue with a description!

---





## CHAPTER 2

---

# The Open Distribution Server

---

The ODS is a Flask application and Celery worker that connect to a Redis server (for worker queues), a database server (SQLite and MySQL currently supported with MSSQL and PostgreSQL planned), and fronted by a web server (Nginx or Apache).

While single-server installers are planned for Linux (Ubuntu/RHEL), Windows, and macOS, the ODS application can be deployed in many different models, such as the application and worker server connecting to remote Redis and database servers, or running as a containerized server in Docker or Kubernetes.

---

**Note:** See the [Docker documentation](#) to read more about using the included `docker-compose.yml` example to launch a development/test instance.

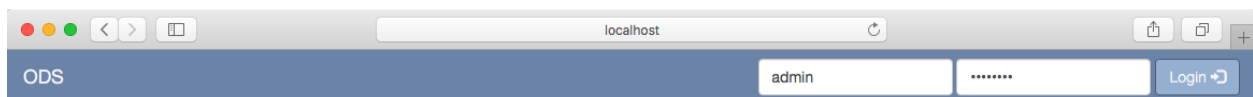
---

## 2.1 Initial Setup

### 2.1.1 Login

During first time initialization, the ODS application will set an initial administrator user account to access the web UI and Admin API.

Username: *admin* Password: *ods1234!*



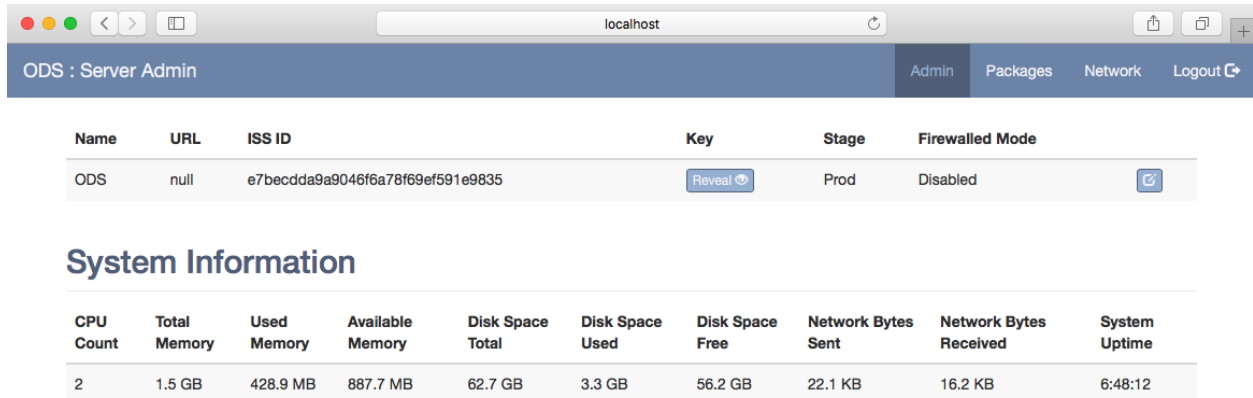
Welcome to the ODS

---

**Warning:** You will be able to change this account, and add others with restricted permissions, in a later update.

## 2.1.2 Server Administration

Under the *Admin* tab you can view the current server settings and information on the system's resources and activities.



The screenshot shows a web browser window with the address bar set to 'localhost'. The page title is 'ODS : Server Admin'. There is a navigation bar with tabs: 'Admin' (selected), 'Packages', 'Network', and 'Logout'. Below the navigation bar is a table with the following columns: 'Name', 'URL', 'ISS ID', 'Key', 'Stage', and 'Firewalled Mode'. The table contains one row for 'ODS' with 'null' as the URL, a long alphanumeric string as the ISS ID, a 'Reveal' button next to the Key, 'Prod' as the Stage, and 'Disabled' as the Firewalled Mode. Below this table is a section titled 'System Information' which contains another table with columns for various system metrics.

Name	URL	ISS ID	Key	Stage	Firewalled Mode
ODS	null	e7becdda9a9046f6a78f69ef591e9835	<a href="#">Reveal</a>	Prod	Disabled

CPU Count	Total Memory	Used Memory	Available Memory	Disk Space Total	Disk Space Used	Disk Space Free	Network Bytes Sent	Network Bytes Received	System Uptime
2	1.5 GB	428.9 MB	887.7 MB	62.7 GB	3.3 GB	56.2 GB	22.1 KB	16.2 KB	6:48:12

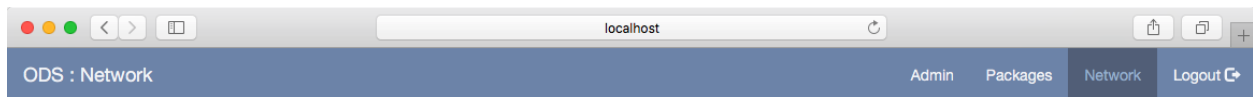
Before you can register the ODS to another ODS you must set the *Name* and *URL*. Click the *Edit* button on the right to open the settings modal to make your modifications.

**Name****URL****Deploy Stage****Firewalled Mode**

## 2.2 ODS Registration

Under the *Network* tab you will have options for registering with other ODS applications. Registration allows ODS instances to communicate with each other and sync files and changes to those files.

To perform the registration you will need the *URL*, *ISS ID*, and *Key* of the remote ODS you are registering with. Enter the values into the provided fields and click *Register*.



### Register with a ODS

URL	<input type="text" value="http://192.168.99.100"/>	ISS ID	<input type="text" value="b56eeb6b-64aa-4382-b8"/>	Key	<input type="text" value="*****"/>	<input type="button" value="Register"/>
-----	--	--------	--	-----	------------------------------------	---

Upon a successful registration you will see the remote ODS appear in the list of registered instances. If not, an error message banner will be displayed on the page.

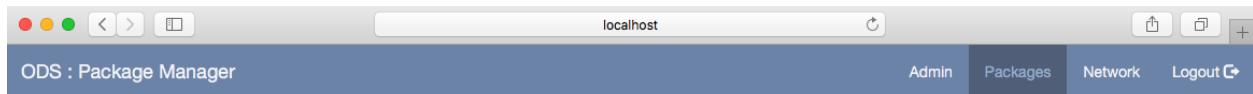
## Registered ODSs

Name	URL	Date Registered	Stage	Firewalled Mode
ODS 1	http://192.168.99.100/	2017-11-09 10:48 PM	Prod	Disabled

## 2.3 Package Management

Under the *Packages* tab you will be able to upload files to the ODS to make available for download. If you have registered with other ODS instances they will receive notifications to sync the package.

Click the *Select File* button and browse for the file to upload. Select a *Stage* to associate with the file and click *Upload*. The *Upload* button will become grayed out during this process.



### Upload a Package

Select File NoMAD.pkg Prod Upload

Upon a successful upload the page will reload and display the new file.

### Available Packages

Filename	Size	SHA1 Hash	State	Stage
NoMAD.pkg	3.3 MB	3fab53c6f12e3d4621b17f728e9b3c522bb90816	Public	Prod

If the server is syncing a file from another ODS it will appear in this list with a *Downloading* status. Upon completion this will change to display *Public* meaning the file is available for download.

## 2.4 Known Issues

**Note:** The ODST Project is currently in an Alpha state. Known issues as they are reported by users and admins testing the iterative builds that won't be immediately addressed will be recorded here with links to the GitHub issue and information on what is planned to address them.

### 2.4.1 File Syncing

- There is likely an issue with the concurrent connections to the database causing the ORM to throw *InternalError* and *InterfaceError* exceptions as it attempts to handle multiple simultaneous requests and tasks. File sync operations currently are not performing adequate error handling to prevent the data from entering an invalid state in the event an error occurs, nor are there any mechanisms in place to initiate an automatic retry. Those items, as well as vastly improved logging, will be implemented to begin addressing these issues. (discovered during testing)

- It has been reported that upload operations can consume up to 3x the disk space of the uploaded file until complete. (reported in Slack #odst)

## 2.5 Admin API

### 2.5.1 Endpoints

Resource	Operation	Description
About	<i>POST /api/admin/about/update</i>	Updates ODS application settings.
	<i>GET /api/admin/about</i>	Returns ODS application settings.
ODS	<i>GET /api/admin/registered_ods</i>	Returns a list of all registered ODS instances.
	<i>POST /api/admin/register</i>	Register with another ODS instance.
Packages	<i>GET /api/admin/packages</i>	Returns a list of all packages on the server.
	<i>GET /api/admin/packages/(package_id_or_name)</i>	Returns a single package by the database ID or
System	<i>GET /api/admin/system</i>	Returns system information.
Upload	<i>POST /api/admin/upload</i>	Upload a file to the ODS.

### 2.5.2 Reference

#### GET /api/admin/about

Returns ODS application settings.

##### Example Request:

```
GET /api/admin/about HTTP/1.1
Accept: application/json
```

##### Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "firewalled_mode": false,
  "iss": "e7becdda9a9046f6a78f69ef591e9835",
  "key": "r3Yt354eDTY0JkaiObpsM4krfkDzdZD9NNYwDr9aSk0=",
  "name": "Example ODS",
  "stage": "Prod",
  "url": "http://192.168.99.100"
}
```

#### POST /api/admin/about/update

Updates ODS application information and settings.

---

**Note:** You cannot update the iss or key values!

---

Accepted values for stage are: Dev, Test, Prod

Accepted values for firewalled\_mode are: Enabled, Disabled

##### Example Request:

```
POST /api/admin/about HTTP/1.1
Content-Type: application/json

{
  "name": "Example ODS",
  "url": "http://192.168.99.100",
  "firewalled_mode": "Disabled",
  "stage": "Prod"
}
```

#### Example Response:

```
HTTP/1.1 201 OK
Content-Type: text/html
```

#### GET /api/admin/packages

Returns a list of all packages on the server. The package objects are nested under an items key.

created value is in ISO 8601 format (without microseconds).

created\_ts value is a Unix timestamp (time since Epoch).

file\_size value is represented in total bytes.

file\_size\_hr is a human readable representation.

#### Example Request:

```
GET /api/admin/packages HTTP/1.1
Accept: application/json
```

#### Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [
    {
      "created": "2017-11-11T03:15:54",
      "created_ts": 1510370154,
      "file_size": 3438044,
      "file_size_hr": "3.3 MB",
      "filename": "NoMAD.pkg",
      "id": 1,
      "sha1": "3fab53c6f12e3d4621b17f728e9b3c522bb90816",
      "stage": "Prod",
      "status": "Public",
      "uuid": "28f7adde4f674873807a4c8c69b641d0"
    }
  ]
}
```

#### GET /api/admin/packages/ (package\_id\_or\_name)

Returns a single package by the database ID or filename.

created value is in ISO 8601 format (without microseconds).

created\_ts value is a Unix timestamp (time since Epoch).

file\_size value is represented in total bytes.

`file_size_hr` is a human readable representation.

`chunks` is an array of indexed hashes for each 1 MB block of the file.

#### Example Request:

```
GET /api/admin/packages/1 HTTP/1.1
Accept: application/json
```

```
GET /api/admin/packages/NoMAD.pkg HTTP/1.1
Accept: application/json
```

#### Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "created": "2017-11-11T03:15:54",
  "created_ts": 1510370154,
  "file_size": 3438044,
  "file_size_hr": "3.3 MB",
  "filename": "NoMAD.pkg",
  "id": 1,
  "sha1": "3fab53c6f12e3d4621b17f728e9b3c522bb90816",
  "stage": "Prod",
  "status": "Public",
  "uuid": "28f7adde4f674873807a4c8c69b641d0"
  "chunks": [
    {
      "index": 0,
      "sha1": "f34fef9be0364d92424106c19596d3fcd1676635"
    },
    {
      "index": 1,
      "sha1": "93633f00f71d3b5494b23b74b37777bc07f9d713"
    },
    {
      "index": 2,
      "sha1": "9e0ca33c872fb9b5d7aa46fbd865c309db792c12"
    },
    {
      "index": 3,
      "sha1": "468521469da7fd40ee72f18e70bf578f055b3878"
    }
  ],
}
```

#### POST /api/admin/register

Register with another ODS instance.

#### Example Request:

```
POST /api/admin/register HTTP/1.1
Content-Type: application/json

{
  "url": "http://192.168.99.101",
  "iss_id": "e7becdda9a9046f6a78f69ef591e9835",
}
```

```
"key": "r3Yt354eDTY0JkaiObpsM4krfkDzdZD9NNYwDr9aSk0="
}
```

**Example Response:**

```
HTTP/1.1 201 OK
Content-Type: text/html
```

**GET /api/admin/registered\_ods**

Returns a list of all registered ODS instances. The `ods` server objects are nested under an `items` key.

`registered_on` value is in ISO 8601 format (without microseconds).

`registered_on_ts` value is a Unix timestamp (time since Epoch).

**Example Request:**

```
GET /api/admin/registered_ods HTTP/1.1
Accept: application/json
```

**Example Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [
    {
      "firewalled_mode": false,
      "id": 1,
      "name": "Example ODS 2",
      "registered_on": "2017-11-10T04:45:47",
      "registered_on_ts": 1510289147,
      "stage": "Prod",
      "url": "http://192.168.99.101"
    }
  ]
}
```

**GET /api/admin/system**

Returns system information.

`disk_`, `mem_`, and `network_` values are represented in total bytes.

`uptime` is represented in total seconds.

Human readable values are denoted by the `_hr` suffix.

**Example Request:**

```
GET /api/admin/system HTTP/1.1
Accept: application/json
```

**Example Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "cpu_count": 2,
```



```

"disk_free": 60352081920,
"disk_free_hr": "56.2 GB",
"disk_total": 67371577344,
"disk_total_hr": "62.7 GB",
"disk_used": 3566796800,
"disk_used_hr": "3.3 GB",
"mem_available": 936255488,
"mem_available_hr": "892.9 MB",
"mem_total": 1567678464,
"mem_total_hr": "1.5 GB",
"mem_used": 444342272,
"mem_used_hr": "423.8 MB",
"network_bytes_received": 54346,
"network_bytes_received_hr": "53.1 KB",
"network_bytes_sent": 45694,
"network_bytes_sent_hr": "44.6 KB",
"uptime": 30806,
"uptime_hr": "8:33:26"
}

```

**POST /api/admin/upload**

Upload a file to the ODS.

Accepted values for stage are: Dev, Test, Prod

**Example Request:**

```

POST /api/admin/register HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryvvcON8g8Lh1D16BS

-----WebKitFormBoundaryvvcON8g8Lh1D16BS
Content-Disposition: form-data; name="file"; filename="NoMAD.pkg"
Content-Type: application/octet-stream

-----WebKitFormBoundaryvvcON8g8Lh1D16BS
Content-Disposition: form-data; name="stage"

Prod
-----WebKitFormBoundaryvvcON8g8Lh1D16BS--

```

**Example Response:**

```

HTTP/1.1 201 OK
Content-Type: application/json

{
  "filename": "NoMAD.pkg",
  "sha1": "3fab53c6f12e3d4621b17f728e9b3c522bb90816"
}

```

## 2.6 The Python Environment

In a custom installation of the ODS application, the configuration of your Python environment may depend upon where and/or how you are doing the deployment.

In a *containerized* or similar deployment, you may install the package dependencies directly to the provided system Python or the one that is installed as a part of building the *container*.

If deploying to a VM or other server, it would be considered best practice to create a Python virtual environment separate from the system's Python and install the package dependencies there.

### 2.6.1 Pipenv

The ODST Project uses *Pipenv* for managing the project dependencies. *Pipenv* uses *Pipfiles* for managing project dependencies and is the officially recommended Python packaging tool from [Python.org](https://python.org).

You can learn more about the *Pipfile* specification on the Python Packaging Authority's GitHub page for the project at <https://github.com/pypa/pipfile>.

If you are new to using *Pipenv* please read the documentation available at <https://docs.pipenv.org/>.

### 2.6.2 Installing Package Dependencies

Install *Pipenv* using *pip*:

```
pip install pipenv
```

To install the project's package dependencies to the system Python run:

```
pipenv install --system
```

To create a virtual environment for the project, run the following commands to build the virtual environment, install the packages, and read back the virtual environment's path:

```
pipenv install --two
pipenv --venv
```

---

**Note:** If you are building a development environment for running tests and building documentation, you must run the following command to also install the development packages:

```
pipenv install --dev
```

---

## 2.7 Application Configuration

The ODS application has variable requirements depending upon the environment.

The application code is contained within the `ods` directory. When deploying with a WSGI server you can use the `application.py` file and the `application` object within it for your WSGI configuration.

**Warning:** The ODS APIs use token authentication and your production ODS instances should use TLS encryption for all traffic!

---

**Note:** An example WSGI configuration file for deploying the ODS application with uWSGI can be found in `/docker/webapp/web-app.ini`.

---

There are a number of options available for deploying the ODS application. See <http://flask.pocoo.org/docs/0.12/deploying/> for more information.

In a single server installation (a standard, minimal install), the ODS is capable of using a local SQLite database for server data. ODS also supports connecting to a MySQL server. MySQL can be running locally on the same server as the ODS application or remotely.

---

**Note:** PostgreSQL and MSSQL support are planned for future support.

---

The ODS uses Redis for queuing tasks and Celery for processing those tasks. Redis can be running locally on the same server as the ODS application or remotely.

To start the Celery worker run the following command from the application directory:

```
celery worker --app ods_worker.celery --workdir /path/to/ODS-dir/
```

## 2.7.1 Environment Variables

The ODS application will read environment variables from the system to configure itself at runtime.

Required Env Vars	Description
ODS_CONF	The path to a configuration file written in Python that contains any of the described environment variables listed here. It is recommended that your SECRET_KEY and DATABASE_KEY be populated using this file.
SECRET_KEY	A cryptographically random key used to secure user sessions.
DATABASE_KEY	A cryptographically random 32-byte key used to encrypt sensitive data within the ODS database.
CELERY_BROKER_URL	The URL to the Redis server. If not provided the value will default to <code>redis://localhost:6379</code> .
CELERY_BACKEND	(optional) An alternative URL to the Redis backend. If not provided it will default to the value of CELERY_BROKER_URL.
UPLOAD_STAGING_DIRECTORY	(optional) You may specify the staging directory where file uploads are cached. If not provided, a randomized temp directory will be created.
DEBUG	(optional) Runs the server in Debug mode providing additional logging output.

The following code will generate a randomized 32-byte key:

```
>> import os
>>> os.urandom(32)
```

## 2.7.2 Database Configuration

By default, the ODS will create a local SQLite database located within the application's directory on the system. To use a database server, set the appropriate environment variables as shown below.

MySQL Server Configuration	
MYSQL_SERVER	The URL to the MySQL server (with port).
MYSQL_DATABASE	The name of the MySQL database to connect to.
MYSQL_USER	The user to authenticate to the database.
MYSQL_PASSWORD	The password to the user.

## 2.8 Example Docker Compose Setup

---

**Note:** This option as provided is primarily meant to serve as a development and testing environment.

---

You can create a full ODS instance using the provided `docker-compose.yml` file on a running Docker host. This Docker Compose configuration will create and launch the following containers on your host:

- Nginx
- MySQL
- Redis
- ODS Web App
- ODS Worker

The containers are isolated by three networks: `proxy` connects Nginx (the externally accessible service) to the ODS web app. The `db` network connects MySQL to both the web app and the worker, and the same is true for the `cache` network. Nginx cannot contact the MySQL, Redis, or worker containers in this configuration.

There will be two data volumes for persisting the MySQL database as well as the file share directory located at `/opt/odst/ods/static/share`. The file share volume is shared between the ODS application and Nginx containers. In this configuration Nginx takes over for serving the packages.

Use the following commands to launch the containers on a Docker host from the ODST repository's directory:

```
docker-compose build
docker-compose up -d
```

Navigate to the IP address of your Docker host in a web browser to begin using the ODS web UI.

---

**Note:** Launch the containers on multiple Docker hosts to test syncing features.

---

---

### Completed Features and Notes

---

- Admin Web UI Logins
  - Default username: `admin`
  - Default password: `ods1234!`
- Server Admin page (UI)
- Package Management (UI)
- **ODS Network (UI)** ODS registration is functional, instances can make API requests to each other, and one-way syncing is implemented.
- **Admin API Endpoints**
  - **GET /api/admin/about** Returns application information.
  - **GET /api/admin/system** Returns system information.
  - **GET /api/admin/packages** Returns all files that have been uploaded.
  - **POST /api/admin/packages** Upload a file to the server. The content-type must be `multipart/form-data` containing `file` and `stage` attributes.



---

### Contributing to ODST

---

The ODST project originated with the Mac Admins community. If you have not, please join the [Mac Admins Slack](#) and join the [#odst channel](#) to join the conversation. Feedback and putting up feature requests for discussion are entirely welcome and desired!

You can also contribute to the development of the ODST project in any of the following ways:

- **Build Testing** Run the latest version as features are committed and test them. Submit issues, provide logs, and provide details on the application deployment. Testing and submitting issues will ensure quality!
- **Optimal Configurations** Administrators and developers experienced with Redis, Nginx, Apache, MySQL, or any of the other technologies that comprise the ODST stack, can help define the baseline configuration of these services for documentation and to be set as a part of the planned installers.
- **Installers** The ODS application can be custom setup for almost any kind of deployment, but an easy option where an administrator can run an installer on a single server is a goal for the project. If you have experience building installers on any platform please reach out!
- **Documentation** Read and scrutinize this documentation. Feel free to make a pull request to correct errors - spelling and grammatical - and incorrect information.





---

## HTTP Routing Table

---

### /api

GET /api/admin/about, 9  
GET /api/admin/packages, 10  
GET /api/admin/packages/(package\_id\_or\_name),  
10  
GET /api/admin/registered\_ods, 12  
GET /api/admin/system, 12  
POST /api/admin/about/update, 9  
POST /api/admin/register, 11  
POST /api/admin/upload, 13