

Installing DSpace

- [Installation Overview](#)
- [Installing the Backend \(Server API\)](#)
 - [Backend Requirements](#)
 - [Backend Installation](#)
- [Installing the Frontend \(User Interface\)](#)
 - [Frontend Requirements](#)
 - [Frontend Installation](#)
- [What Next?](#)
- [Common Installation Issues](#)
 - [Troubleshoot an error or find detailed error messages](#)
 - [User Interface never appears \(no content appears\) or "Proxy server received an invalid response"](#)
 - [User Interface partially load but then spins \(never fully loads or some content doesn't load\)](#)
 - ["500 Service Unavailable" from the User Interface](#)
 - ["No _links section found at..." error from User Interface](#)
 - ["RangeError: Maximum call stack size exceeded"](#)
 - ["XMLHttpRequest.. has been blocked by CORS policy" or "CORS error" or "Invalid CORS request"](#)
 - [Cannot login from the User Interface with a password that I know is valid](#)
 - ["403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token"](#)
 - [Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend](#)
 - [My REST API is running under HTTPS, but some of its "link" URLs are switching to HTTP](#)
 - [My User Interface's robots.txt has incorrect sitemap URLs](#)
 - [Cannot upload file from User Interface](#)
 - [Javascript heap out of memory](#)
 - [Solr responds with "Expected mime type application/octet-stream but got text/html" \(404 Not Found\)](#)
 - [Database errors occur when you run ant fresh_install](#)

Installation Overview

Try out DSpace 7 before you install

If you'd like to quickly try out DSpace 7 before a full installation, see [Try out DSpace 7](#) for instructions on a quick install via Docker.

As of version 7 (and above), the DSpace application is split into a "frontend" (User Interface) and a "backend" (Server API). Most institutions will want to install BOTH. However, you can decide whether to run them on the same machine or separate machines.

- The DSpace Frontend consists of a User Interface built on [Angular.io](#). It is a Node.js web application, i.e. once it is built/compiled, it only require Node.js to run. It cannot be run "standalone", as it *requires* a valid DSpace Backend to function. The frontend provides all user-facing functionality.
- The DSpace Backend consists of a Server API ("server" webapp), built on [Spring Boot](#). It is a Java web application. It can be run standalone, however it has no user interface. The backend provides all machine-based interfaces, including the REST API, OAI-PMH, SWORD (v1 and v2) and RDF.

We recommend installing the Backend **first**, as the Frontend requires a valid Backend to run properly.

Installing the Backend (Server API)

Backend Requirements

- [UNIX-like OS or Microsoft Windows](#)
- [Java JDK 11 or 17 \(OpenJDK or Oracle JDK\)](#)
- [Apache Maven 3.5.4 or above \(Java build tool\)](#)
 - [Configuring a Maven Proxy](#)
- [Apache Ant 1.10.x or later \(Java build tool\)](#)
- [Relational Database \(PostgreSQL\)](#)
 - [PostgreSQL 12.x, 13.x, 14.x or 15.x \(with pgcrypto installed\)](#)
 - [Oracle \(UNSUPPORTED AS OF 7.6\)](#)
- [Apache Solr 8.x \(full-text index/search service\)](#)
- [Servlet Engine \(Apache Tomcat 9, Jetty, Caucho Resin or equivalent\)](#)
- [\(Optional\) IP to City Database for Location-based Statistics](#)

UNIX-like OS or Microsoft Windows


- UNIX-like operating system (Linux, HP/UX, Mac OSX, etc.) : Many distributions of Linux/Unix come with some of the dependencies below pre-installed or easily installed via updates. You should consult your particular distribution's documentation or local system administrators to determine what is already available.
- Microsoft Windows: While DSpace can be run on Windows servers, most institutions tend to run it on a UNIX-like operating system.

Java JDK 11 or 17 (OpenJDK or Oracle JDK)


JDK17 support was first available in DSpace 7.2. DSpace 7.1 and 7.0 only supported JDK 11.

- OpenJDK download and installation instructions can be found here <http://openjdk.java.net/install/>. Most operating systems provide an easy path to install OpenJDK. Just be sure to install the full JDK (development kit), and not the JRE (which is often the default example).
- Oracle's Java can be downloaded from the following location: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Make sure to download the appropriate version of the Java SE JDK.

Make sure to install the JDK and not just the JRE

 DSpace requires the full JDK (Java Development Kit) be installed, rather than just the JRE (Java Runtime Environment). So, please be sure that you are installing the full JDK and not just the JRE.

Only JDK11 and JDK 17 are fully supported

 Older versions of Java are unsupported. This includes JDK v7-10.

Newer versions of Java may work (e.g. JDK v12-16), but we do not recommend running them in Production. We highly recommend running only Java LTS (Long Term Support) releases in Production, as non-LTS releases may not receive ongoing security fixes. As of this DSpace release, JDK11 and JDK 17 are the two most recent Java LTS releases. As soon as the next Java LTS release is available, we will analyze it for compatibility with this release of DSpace. For more information on Java releases, see the Java roadmaps for [Oracle](#) and/or [OpenJDK](#).

Apache Maven 3.5.4 or above (Java build tool)

We recommend using the most recent version of Maven that you can, as newer releases may include performance improvements and security updates. We recommend avoiding any that are "end of life" per <https://maven.apache.org/docs/history.html>

Maven is necessary in the first stage of the build process to assemble the installation package for your DSpace instance. It gives you the flexibility to customize DSpace using the existing Maven projects found in the `[dspace-source]/dspace/modules` directory or by adding in your own Maven project to build the installation package for DSpace, and apply any custom interface "overlay" changes.

Maven can be downloaded from <http://maven.apache.org/download.html> It is also provided via many operating system package managers.


Configuring a Maven Proxy

You can configure a proxy to use for some or all of your HTTP requests in Maven. The username and password are only required if your proxy requires basic authentication (note that later releases may support storing your passwords in a secured keystore, in the meantime, please ensure your `settings.xml` file (usually `$(user.home)/.m2/settings.xml`) is secured with permissions appropriate for your operating system).

Example:

```
<settings>
.
.
<proxies>
  <proxy>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.somewhere.com</host>
    <port>8080</port>
    <username>proxyuser</username>
    <password>somepassword</password>
    <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
  </proxy>
</proxies>
.
.
</settings>
```

Apache Ant 1.10.x or later (Java build tool)

 While Apache Ant recommends using v1.10.x for Java 11, we've also had some success with recent versions of 1.9.x (specifically v1.9.15 seems to work fine with Java 11). That said, earlier versions of v1.9.x are not compatible with Java 11.

Apache Ant is required for the second stage of the build process (deploying/installing the application). First, Maven is used to construct the installer (`[dspace-source]/dspace/target/dspace-installer`), after which Ant is used to install/deploy DSpace to the installation directory.

Ant can be downloaded from the following location: <http://ant.apache.org> It is also provided via many operating system package managers.

Relational Database (PostgreSQL)

PostgreSQL 12.x, 13.x, 14.x or 15.x (with pgcrypto installed)

- PostgreSQL can be downloaded from <http://www.postgresql.org/>. It is also provided via many operating system package managers.
 - Make sure to select a version of PostgreSQL that is still [under support from the PostgreSQL team](#).
 - If the version of Postgres provided by your package manager is outdated, you may wish to use one of the official PostgreSQL provided repositories:
 - Linux users can select their OS of choice for detailed instructions on using the official PostgreSQL apt or yum repository: <http://www.postgresql.org/download/linux/>
 - Windows users will need to use the windows installer: <http://www.postgresql.org/download/windows/>
 - Mac OSX users can choose their preferred installation method: <http://www.postgresql.org/download/macosx/>
- Install the [pgcrypto extension](#). It will also need to be enabled on your DSpace Database (see Installation instructions below for more info). The pgcrypto extension allows DSpace to create UUIDs (universally unique identifiers) for all objects in DSpace, which means that (internal) object identifiers are now globally unique and no longer tied to database sequences.
 - On most Linux operating systems (Ubuntu, Debian, RedHat), this extension is provided in the "postgresql-contrib" package in your package manager. So, ensure you've installed "postgresql-contrib".
 - On Windows, this extension should be provided automatically by the installer (check your "[PostgreSQL]/share/extension" folder for files starting with "pgcrypto")
- Unicode (specifically UTF-8) support must be enabled (but this is enabled by default).
- Once installed, you need to enable TCP/IP connections (DSpace uses JDBC):
 - In `postgresql.conf`: uncomment the line starting: `listen_addresses = 'localhost'`. This is the default, in recent PostgreSQL releases, but you should at least check it.
 - Then tighten up security a bit by editing `pg_hba.conf` and adding this line:

```
host dspace dspace 127.0.0.1 255.255.255.255 md5
```

This should appear *before* any lines matching all databases, because the first matching rule governs.

- Then restart PostgreSQL.

Oracle (UNSUPPORTED AS OF 7.6)

As of the 7.6 release, Oracle databases are no longer supported

Oracle support has been removed as was [previously announced in March 2022](#) on our mailing lists. See <https://github.com/DSpace/DSpace/issues/8214>

All DSpace sites should now use PostgreSQL (see above)

- Details on acquiring Oracle can be downloaded from the following location: <http://www.oracle.com/database/>. You will need to create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is suggested that the Oracle database use the same character set. You will also need to create a user account for DSpace (e.g. `dspace`) and ensure that it has permissions to add and remove tables in the database. Refer to the Quick Installation for more details.
 - **NOTE:** If the database server is not on the same machine as DSpace, you must install the Oracle client to the DSpace server and point to `nsnames.ora` and `listener.ora` files to the database the Oracle server.

Apache Solr 8.x (full-text index/search service)

Solr 8.11.1 or above is recommended as all prior 8.x releases are vulnerable to CVE-2021-44228 (log4j critical vulnerability). If you must use a prior version of 8.x, make sure to add `"-Dlog4j2.formatMsgNoLookups=true"` to your SOLR_OPTS environment variable, see <https://solr.apache.org/security.html#apache-solr-affected-by-apache-log4j-cve-2021-44228>

Solr 9 is not yet fully supported, but can be used provided that you make minor modification to the out-of-the-box `"search/conf/solrconfig.xml"` that comes with DSpace 7. See [this below comment](#) for details.

Make sure to install Solr with Authentication disabled (which is the default). DSpace does not yet support authentication to Solr (see <https://github.com/DSpace/DSpace/issues/3169>). Instead, we recommend placing Solr behind a firewall and/or ensuring port 8983 (which Solr runs on) is not available for public/anonymous access on the web. Solr only needs to be accessible to requests from the DSpace backend.

Solr can be obtained at [the Apache Software Foundation site for Solr](#). You may wish to read portions of [the quick-start tutorial](#) to make yourself familiar with Solr's layout and operation. Unpack a Solr .tgz or .zip archive in a place where you keep software that is not handled by your operating system's package management tools, and arrange to have it running whenever DSpace is running. You should ensure that Solr's index directories will have plenty of room to grow. You should also ensure that port 8983 is not in use by something else, or configure Solr to use a different port.

If you are looking for a good place to put Solr, consider `/opt` or `/usr/local`. You can simply unpack Solr in one place and use it. Or you can configure Solr to keep its indexes elsewhere, if you need to – see the Solr documentation for how to do this.

It is not necessary to dedicate a Solr instance to DSpace, if you already have one and want to use it. Simply copy DSpace's cores to a place where they will be discovered by Solr. See below.

Servlet Engine (Apache Tomcat 9, Jetty, Caucho Resin or equivalent)

Only Tomcat 9 is supported at this time. Tomcat 10 is incompatible with Tomcat 9 and will not be supported until DSpace 8.0 at the earliest. See <https://github.com/DSpace/DSpace/issues/8713> for more details

- **Apache Tomcat 9.** Tomcat can be downloaded from the following location: <http://tomcat.apache.org>. It is also provided via many operating system package managers.
 - *The Tomcat owner* (i.e. the user that Tomcat runs as) **must have read/write access to the DSpace installation directory** (i.e. `[dspace]`). There are a few common ways this may be achieved:
 - One option is to specifically give the Tomcat user (often named "tomcat") ownership of the `[dspace]` directories, for example:

```
# Change [dspace] and all subfolders to be owned by "tomcat"
chown -R tomcat:tomcat [dspace]
```

- Another option is to have Tomcat itself *run* as a new user named "dspace" (see installation instructions below). Some operating systems make modifying the Tomcat "run as" user easily modifiable via an environment variable named TOMCAT_USER. This option may be more desirable if you have multiple Tomcat instances running, and you do not want all of them to run under the same Tomcat owner.
- On *Debian* systems, you may also need to modify or override the "tomcat.service" file to specify the DSpace installation directory in the list of ReadWritePaths. For example:

```
# Replace [dspace] with the full path of your DSpace install
ReadWritePaths=[dspace]
```

- You need to ensure that Tomcat a) has enough memory to run DSpace, and b) uses UTF-8 as its default file encoding for international character support. So ensure in your startup scripts (etc) that the following environment variable is set: `JAVA_OPTS="-Xmx512M -Xms64M -Dfile.encoding=UTF-8"`
- **Modifications in `[tomcat]/conf/server.xml`:** You also need to alter Tomcat's default configuration to support searching and browsing of multi-byte UTF-8 correctly. You need to add a configuration option to the `<Connector>` element in `[tomcat]/conf/server.xml`: `URIEncoding="UTF-8"` e.g. if you're using the default Tomcat config, it should read:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080"
            minSpareThreads="25"
            enableLookups="false"
            redirectPort="8443"
            connectionTimeout="20000"
            disableUploadTimeout="true"
            URIEncoding="UTF-8" />
```

You may change the port from 8080 by editing it in the file above, and by setting the variable `CONNECTOR_PORT` in `server.xml`. You should set the `URIEncoding` even if you are running Tomcat behind a reverse proxy (Apache HTTPD, Nginx, etc.) via AJP.

- **Jetty or Caucho Resin**

- DSpace 7 has not been tested with Jetty or Caucho Resin, after the switch to Java 11
 - Older versions of DSpace were able to run on a Tomcat-equivalent servlet Engine, such as Jetty (<https://www.eclipse.org/jetty/>) or Caucho Resin (<http://www.caucho.com/>). If you choose to use a different servlet container, please ensure that it supports Servlet Spec 3.1 (or above).
 - Jetty and Resin are configured for correct handling of UTF-8 by default.

(Optional) IP to City Database for Location-based Statistics

Optionally, if you wish to record the geographic locations of clients in DSpace usage statistics records, you will need to install (and regularly update) one of the following:

- Either, a copy of [MaxMind's GeoLite City database](#) (in MMDB format)
 - NOTE: Installing MaxMind GeoLite2 is *free*. However, you **must** sign up for a (free) MaxMind account in order to obtain a license key to use the GeoLite2 database.
 - You may download GeoLite2 directly from MaxMind, or many Linux distributions provide the `geoipupdate` tool directly via their package manager. You will still need to configure your license key prior to usage.
 - Once the "GeoLite2-City.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
 - See the "Managing the City Database File" section of [SOLR Statistics](#) for more information about using a City Database with DSpace.
- Or, you can alternatively use/install [DB-IP's City Lite database](#) (in MMDB format)
 - This database is also free to use, but does **not** require an account to download.
 - Once the "dbip-city-lite.mmdb" database file is installed on your system, you will need to configure its location as the value of `usage-statistics.dbfile` in your `local.cfg` configuration file.
 - See the "Managing the City Database File" section of [SOLR Statistics](#) for more information about using a City Database with DSpace.

Backend Installation

1. Install all the [Backend Requirements](#) listed above.
2. **Create a DSpace operating system user (optional)**. As noted in the prerequisites above, Tomcat (or Jetty, etc) **must run as** an operating system user account that has full read/write access to the DSpace installation directory (i.e. `[dspace]`). Either you must ensure the Tomcat owner also owns `[dspace]`, OR you can create a new "dspace" user account, and ensure that Tomcat also runs as that account:

```
useradd -m dspace
```

The choice that makes the most sense for you will probably depend on how you installed your servlet container (Tomcat/Jetty/etc). If you installed it from source, you will need to create a user account to run it, and that account can be named anything, e.g. 'dspace'. If you used your operating system's package manager to install the container, then a user account should have been created as part of that process and it will be much easier to use that account than to try to change it.

3. **Download the latest DSpace release** from the DSpace GitHub Repository. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. `dspace-7.2`) or branch.
4. **Unpack the DSpace software**. After downloading the software, based on the compression file format, choose one of the following methods to unpack your software:

- a. **Zip file.** If you downloaded *dspace-7.2.zip* do the following:

```
unzip dspace-7.2.zip
```

- b. **.gz file.** If you downloaded *dspace-7.2.tar.gz* do the following:

```
gunzip -c dspace-7.2.tar.gz | tar -xf -
```

For ease of reference, we will refer to the location of this unzipped version of the DSpace release as *[dspace-source]* in the remainder of these instructions. After unpacking the file, the user may wish to change the ownership of the dspace-7.x folder to the "dspace" user. (And you may need to change the group).

5. Database Setup

- **PostgreSQL:**

- Create a dspace database user (this user can have any name, but we'll assume you name it "dspace"). This is entirely separate from the dspace operating-system user created above:

```
createuser --username=postgres --no-superuser --pwprompt dspace
```

You will be prompted (twice) for a password for the new dspace user. Then you'll be prompted for the password of the PostgreSQL superuser (postgres).

- Create a dspace database, owned by the dspace PostgreSQL user. Similar to the previous step, this can only be done by a "superuser" account in PostgreSQL (e.g. postgres):

```
createdb --username=postgres --owner=dspace --encoding=UNICODE dspace
```

You will be prompted for the password of the PostgreSQL superuser (postgres).

- Finally, you **MUST** enable the [pgcrypto extension](#) on your new dspace database. Again, this can only be enabled by a "superuser" account (e.g. postgres)

```
# Login to the database as a superuser, and enable the pgcrypto extension on this database
psql --username=postgres dspace -c "CREATE EXTENSION pgcrypto;"
```

The "CREATE EXTENSION" command should return with no result if it succeeds. If it fails or throws an error, it is likely you are missing the required pgcrypto extension (see [Database Prerequisites](#) above).

- **Alternative method: How to enable pgcrypto via a separate database schema.** While the above method of enabling pgcrypto is perfectly fine for the majority of users, there may be some scenarios where a database administrator would prefer to install extensions into a database schema that is *separate from* the DSpace tables. Developers also may wish to install pgcrypto into a separate schema if they plan to "clean" (recreate) their development database frequently. Keeping extensions in a separate schema from the DSpace tables will ensure developers would NOT have to continually re-enable the extension each time you run a `./dspace database clean`. If you wish to install pgcrypto in a separate schema here's how to do that:

```
# Login to the database as a superuser
psql --username=postgres dspace
# Create a new schema in this database named "extensions" (or whatever you want to
name it)
CREATE SCHEMA extensions;
# Enable this extension in this new schema
CREATE EXTENSION pgcrypto SCHEMA extensions;
# Grant rights to call functions in the extensions schema to your dspace user
GRANT USAGE ON SCHEMA extensions TO dspace;

# Append "extensions" on the current session's "search_path" (if it doesn't already
exist in search_path)
# The "search_path" config is the list of schemas that Postgres will use
SELECT set_config('search_path',current_setting('search_path') || ',extensions',
false) WHERE current_setting('search_path') !~ '^(^|,|)extensions(,|$)';
# Verify the current session's "search_path" and make sure it's correct
SHOW search_path;
# Now, update the "dspace" Database to use the same "search_path" (for all future
sessions) as we've set for this current session (i.e. via set_config() above)
ALTER DATABASE dspace SET search_path FROM CURRENT;
```

- **Oracle (UNSUPPORTED AS OF 7.6):**

- Setting up DSpace to use Oracle is a bit different now. You will need still need to get a copy of the Oracle JDBC driver, but instead of copying it into a lib directory you will need to install it into your local Maven repository. (You'll need to download it first from this location: <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html>.) Run the following command (all on one line):

```
mvn install:install-file
-Dfile=ojdbc6.jar
-DgroupId=com.oracle
-DartifactId=ojdbc6
-Dversion=11.2.0.4.0
-Dpackaging=jar
-DgeneratePom=true
```

- You need to compile DSpace with an Oracle driver (ojdbc6.jar) corresponding to your Oracle version - update the version in `[dspace-source]/pom.xml` E.g.:

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc6</artifactId>
  <version>11.2.0.4.0</version>
</dependency>
```

- Create a database for DSpace. Make sure that the character set is one of the Unicode character sets. DSpace uses UTF-8 natively, and it is required that the Oracle database use the same character set. Create a user account for DSpace (e.g. `dspace`) and ensure that it has permissions to add and remove tables in the database.
- NOTE: You will need to ensure the proper `db.*` settings are specified in your `local.cfg` file (see next step), as the defaults for all of these settings assuming a PostgreSQL database backend.

```
db.url = jdbc:oracle:thin:@host:port/SID
# e.g. db.url = jdbc:oracle:thin:@//localhost:1521/xe
# NOTE: in db.url, SID is the SID of your database defined in tnsnames.ora
# the default Oracle port is 1521
# You may also use a full SID definition, e.g.
# db.url = jdbc:oracle:thin:@(description=(address_list=(address=(protocol=TCP)
(host=localhost)(port=1521)))(connect_data=(service_name=DSPACE)))

# Oracle driver and dialect
db.driver = oracle.jdbc.OracleDriver
db.dialect = org.hibernate.dialect.Oracle10gDialect

# Specify DB username, password and schema to use
db.username =
db.password =
db.schema = ${db.username}
# For Oracle, schema is equivalent to the username of your database account,
# so this may be set to ${db.username} in most scenarios
```

- Later, during the Maven build step, don't forget to specify `mvn -Ddb.name=oracle` package

6. Initial Configuration (local.cfg):

Create your own `[dspace-source]/dspace/config/local.cfg` configuration file. You may wish to simply copy the provided `[dspace-source]/dspace/config/local.cfg.EXAMPLE`. This `local.cfg` file can be used to store any configuration changes that you wish to make which are local to your installation (see [local.cfg configuration file](#) documentation). ANY setting may be copied into this `local.cfg` file from the `dspace.cfg` or any other `*.cfg` file in order to override the default setting (see note below). For the initial installation of DSpace, there are some key settings you'll likely want to override. Those are provided in the `[dspace-source]/dspace/config/local.cfg.EXAMPLE`. (NOTE: Settings followed with an asterisk (*) are highly recommended, while all others are optional during initial installation and may be customized at a later time.)

- `dspace.dir*` - must be set to the `[dspace]` (installation) directory (NOTE: On Windows be sure to use forward slashes for the directory path! For example: "C:/dspace" is a valid path for Windows.)
- `dspace.server.url*` - complete URL of this DSpace backend (including port and any subpath). **Do not end with '/'**. For example: `http://localhost:8080/server`
- `dspace.ui.url*` - complete URL of the DSpace frontend (including port and any subpath). REQUIRED for the REST API to fully trust requests from the DSpace frontend. **Do not end with '/'**. For example: `http://localhost:4000`
- `dspace.name` - Human-readable, "proper" name of your server, e.g. "My Digital Library".
- `solr.server*` - complete URL of the Solr server. DSpace makes use of [Solr](#) for indexing purposes. `http://localhost:8983/solr` unless you changed the port or installed Solr on some other host.
- `default.language` - Default language for all metadata values (defaults to "en_US")
- `db.url*` - The full JDBC URL to your database (examples are provided in the `local.cfg.EXAMPLE`)
- `db.driver*` - Which database driver to use for PostgreSQL (default should be fine)
- `db.dialect*` - Which database dialect to use for PostgreSQL (default should be fine)
- `db.username*` - the database username used in the previous step.

- `db.password*` - the database password used in the previous step.
- `db.schema*` - the database schema to use (examples are provided in the `local.cfg.EXAMPLE`)
- `mail.server` - fully-qualified domain name of your outgoing mail server.
- `mail.from.address` - the "From:" address to put on email sent by DSpace.
- `feedback.recipient` - mailbox for feedback mail.
- `mail.admin` - mailbox for DSpace site administrator.
- `alert.recipient` - mailbox for server errors/alerts (not essential but very useful!)
- `registration.notify` - mailbox for emails when new users register (optional)

Your `local.cfg` file can override ANY settings from other `*.cfg` files in DSpace

The provided `local.cfg.EXAMPLE` only includes a small subset of the configuration settings available with DSpace. It provides a good starting point for your own `local.cfg` file.

However, you should be aware that ANY configuration can now be copied into your `local.cfg` to override the default settings. This includes ANY of the settings/configurations in:

- The primary `dspace.cfg` file (`[dspace]/config/dspace.cfg`)
- Any of the module configuration files (`[dspace]/config/modules/*.cfg` files)
- Any of the Spring Boot settings (`[dspace-src]/dspace-server-webapp/src/main/resources/application.properties`)

Individual settings may also be commented out or removed in your `local.cfg`, in order to re-enable default settings.

See the [Configuration Reference](#) section for more details.

7. **DSpace Directory:** Create the directory for the DSpace backend installation (i.e. `[dspace]`). As `root` (or a user with appropriate permissions), run:

```
mkdir [dspace]
chown dspace [dspace]
```

(Assuming the `dspace` UNIX username.)

8. **Build the Installation Package:** As the `dspace` UNIX user, generate the DSpace installation package.

```
cd [dspace-source]
mvn package
```

Building with Oracle Database Support (UNSUPPORTED AS OF 7.6)

Without any extra arguments, the DSpace installation package is initialized for PostgreSQL. If you want to use Oracle instead, you should build the DSpace installation package as follows:

```
mvn -Ddb.name=oracle package
```

9. **Install DSpace Backend:** As the `dspace` UNIX user, install DSpace to `[dspace]`:

```
cd [dspace-source]/dspace/target/dspace-installer
ant fresh_install
```

To see a complete list of build targets, run: `ant help` *The most likely thing to go wrong here is the test of your database connection. See the [Common Installation Issues](#) Section below for more details.*

10. **Initialize your Database:** While this step is optional (as the DSpace database should auto-initialize itself on first startup), it's always good to verify one last time that your database connection is working properly. To initialize the database run:

```
[dspace]/bin/dspace database migrate
```

- a. After running this script, it's a good idea to run `./dspace database info` to check that your database has been fully initialized. A fully initialized database should list the state of all migrations as either "Success" or "Out of Order". If any migrations have failed or are still listed as "Pending", then you need to check your `dspace.log` for possible "ERROR" messages. If any errors appeared, you will need to resolve them before continuing.

11. **Deploy Server web application:** The DSpace backend consists of a single "server" webapp (in `[dspace]/webapps/server`). You need to deploy this webapp into your Servlet Container (e.g. Tomcat). Generally, there are two options (or techniques) which you could use...either configure Tomcat to find the DSpace "server" webapp, or copy the "server" webapp into Tomcat's own webapps folder.
 - *Technique A.* Tell your Tomcat/Jetty/Resin installation where to find your DSpace web application(s). As an example, in the directory `[tomcat]/conf/Catalina/localhost` you could add files similar to the following (but replace `[dspace]` with your installation location):

DEFINE A CONTEXT PATH FOR DSpace Server webapp: server.xml

```
<?xml version='1.0'?>
<Context
    docBase="[dspace]/webapps/server"/>
```

The name of the file (not including the suffix ".xml") will be the name of the context, so for example `server.xml` defines the context at <http://host:8080/server>. To define the *root context* (<http://host:8080/>), name that context's file `ROOT.xml`. Optionally, you can also choose to install the old, deprecated "rest" webapp if you

- **Technique B.** Simple and complete. You copy only (or all) of the DSpace Web application(s) you wish to use from the `[dspace]/webapps` directory to the appropriate directory in your Tomcat/Jetty/Resin installation. For example:
`cp -R [dspace]/webapps/* [tomcat]/webapps` (This will copy all the web applications to Tomcat).
`cp -R [dspace]/webapps/server [tomcat]/webapps` (This will copy only the Server web application to Tomcat.)

To define the *root context* (<http://host:8080/>), name that context's directory `ROOT`.

12. **Optionally, also install the deprecated DSpace 6.x REST API web application.** If you previously used the DSpace 6.x REST API, for backwards compatibility the old, deprecated "rest" webapp is still available to install (in `[dspace]/webapps/rest`). It is NOT used by the DSpace frontend. So, most users should skip this step.
13. **Copy Solr cores:** DSpace installation creates a set of four empty Solr cores already configured.

- a. Copy them from `[dspace]/solr` to the place where your Solr instance will discover them. For example:

```
# [solr] is the location where Solr is installed.
# NOTE: On Debian systems the configsets may be under /var/solr/data/configsets
cp -R [dspace]/solr/* [solr]/server/solr/configsets

# Make sure everything is owned by the system user who owns Solr
# Usually this is a 'solr' user account
# See https://solr.apache.org/guide/8_1/taking-solr-to-production.html#create-the-solr-user
chown -R solr:solr [solr]/server/solr/configsets
```

- b. Start (or re-start) Solr. For example:

```
[solr]/bin/solr restart
```

- c. You can check the status of Solr and your new DSpace cores by using its administrative web interface. Browse to `${solr.server}` (e.g. `http://localhost:8983/solr/`) to see if Solr is running well, then look at the cores by selecting (on the left) Core Admin or using the Core Selector drop list.

- i. For example, to test that your "search" core is setup properly, try accessing the URL `${solr.server}/search/select`. It should run an empty query against the "search" core, returning an empty JSON result. If it returns an error, then that means your "search" core is missing or not installed properly.

14. **Create an Administrator Account:** Create an initial administrator account from the command line:

```
[dspace]/bin/dspace create-administrator
```

15. **Initial Startup!** Now the moment of truth! Start up (or restart) Tomcat/Jetty/Resin.
 - a. *REST API Interface* - (e.g.) <http://dspace.myu.edu:8080/server/>
 - b. *OAI-PMH Interface* - (e.g.) <http://dspace.myu.edu:8080/server/oai/request?verb=Identify>
 - c. For an example of what the default backend looks like, visit the Demo Backend: <https://demo.dspace.org/server/>
16. **Setup scheduled tasks for behind-the-scenes processes:** For all features of DSpace to work properly, there are some scheduled tasks you MUST setup to run on a regular basis. Some examples are tasks that help create thumbnails (for images), do full-text indexing (of textual content) and send out subscription emails. See the [Scheduled Tasks via Cron](#) for more details.
17. **Production Installation (adding HTTPS support):** *Running the DSpace Backend on HTTP & port 8080 is only usable for local development environments (where you are running the UI and REST API from the same machine, and only accessing them via localhost URLs).* **If you want to run DSpace in Production, you MUST run the backend with HTTPS support** (otherwise logins will not work outside of your local domain).
 - a. For HTTPS support, we recommend installing either [Apache HTTPD](#) or [Nginx](#), configuring SSL at that level, and proxying all requests to your Tomcat installation. Keep in mind, if you want to host both the DSpace Backend and Frontend on the same server, you can use one installation of Apache HTTPD or Nginx to manage HTTPS/SSL and proxy to both.
 - b. *Apache HTTPD:* These instructions are specific to Apache HTTPD, but a similar setup can be achieved with Nginx (see below)
 - i. Install [Apache HTTPD](#), e.g. `sudo apt install apache2`
 - ii. Install `mod_headers`, `mod_proxy` and `mod_proxy_ajp` (or `mod_proxy_http`) modules, e.g. `sudo a2enmod headers; sudo a2enmod proxy; sudo a2enmod proxy_ajp`
 1. Alternatively, you can choose to use [mod_proxy_http](#) to create an http proxy. A separate example is commented out below
 - iii. For `mod_proxy_ajp` to communicate with Tomcat, you'll need to enable Tomcat's AJP connector in your Tomcat's `server.xml`:


```
<Connector protocol="AJP/1.3" port="8009" redirectPort="8443" URIEncoding="UTF-8" />
```

- iv. Restart Apache to enable these modules
- v. Obtain an SSL certificate for HTTPS support. If you don't have one yet, you can use Let's Encrypt (for free) using the "certbot" tool: <https://certbot.eff.org/>
- vi. Now, setup a new VirtualHost for your site (using HTTPS / port 443) which proxies all requests to Tomcat's AJP connector (running on port 8009)

```
<VirtualHost _default_:443>
    # Add your domain here. We've added "my.dspace.edu" as an example
    ServerName my.dspace.edu
    .. setup your host how you want, including log settings...      .. setup your host how
you want, including log settings...

    # Most installs will need these options enabled to ensure DSpace knows its hostname and
scheme (http or https)
    # Also required to ensure correct sitemap URLs appear in /robots.txt for User Interface.
    ProxyPreserveHost On
    RequestHeader set X-Forwarded-Proto https

    SSLEngine on
    SSLCertificateFile [full-path-to-PEM-cert]
    SSLCertificateKeyFile [full-path-to-cert-KEY]
    # LetsEncrypt certificates (and possibly others) may require a chain file be specified
    # in order for the UI / Node.js to validate the HTTPS connection.
    #SSLCertificateChainFile [full-path-to-chain-file]

    # Proxy all HTTPS requests to "/server" from Apache to Tomcat via AJP connector
    ProxyPass /server ajp://localhost:8009/server
    ProxyPassReverse /server ajp://localhost:8009/server

    # If you would rather use mod_proxy_http as an http proxy to port 8080
    # then use these settings instead
    #ProxyPass /server http://localhost:8080/server
    #ProxyPassReverse /server http://localhost:8080/server
</VirtualHost>
```

- c. *NGinx*: These instructions are specific to NGinx.
 - i. Install/Setup [NGinx](#)
 - ii. Sample NGinx "server block" configuration. Keep in mind we are only providing basic example settings.

```
# Setup HTTP to redirect to HTTPS
server {
    listen 80;
    # Add your domain here. We've added "my.dspace.edu" as an example
    server_name my.dspace.edu;
    rewrite ^ https://my.dspace.edu permanent;
}

# Setup HTTPS access
server {
    listen 443 ssl;
    # Add your domain here. We've added "my.dspace.edu" as an example
    server_name my.dspace.edu;

    # Add your SSL certificate/key path here
    # NOTE: For LetsEncrypt, the certificate should be the full certificate chain file
    ssl_certificate my.dspace.edu.crt (or PEM);
    ssl_certificate_key my.dspace.edu.key;

    # Proxy all HTTPS requests to "/server" from NGinx to Tomcat on port 8080
    location /server {
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Host $host;
        proxy_pass http://localhost:8080/server;
    }
}
```

- d. After switching to HTTPS, make sure to go back and update the URLs (primarily `dspace.server.url`) in your `local.cfg` to match the new URL of your backend (REST API). This will require briefly rebooting Tomcat.

Installing the Frontend (User Interface)

Below instructions are specific to 7.2 (or later)



The Frontend Instructions below are specific to 7.2 or later. For Frontend Installation instructions for 7.0 or 7.1, see [7.0-7.1 Frontend Installation](#)

Frontend Requirements

- [UNIX-like OS or Microsoft Windows](#)
- [Node.js \(v16.x or v18.x\)](#)
- [Yarn \(v1.x\)](#)
- [PM2 \(or another Process Manager for Node.js apps\)](#) (optional, but recommended for Production)
- [DSpace 7.x Backend \(see above\)](#)

UNIX-like OS or Microsoft Windows

- UNIX-like operating system (Linux, HP/UX, Mac OSX, etc.) : Many distributions of Linux/Unix come with some of the dependencies below pre-installed or easily installed via updates. You should consult your particular distribution's documentation or local system administrators to determine what is already available.
- Microsoft Windows: While DSpace can be run on Windows servers, most institutions tend to run it on a UNIX-like operating system.

Node.js (v16.x or v18.x)

At this time, the DSpace 7 frontend has build issues with Node v20.x. See this bug ticket for more details: <https://github.com/DSpace/dspace-angular/issues/2290>



- Node.js can be found at <https://nodejs.org/>. It may be available through your Linux distribution's package manager. We recommend running a [Long Term Support \(LTS\) version](#) (even numbered releases). Non-LTS versions (odd numbered releases) are not recommended.
- Node.js is a Javascript runtime that also provides [npm](#) (Node Package Manager). It is used to both build and run the frontend.
- **NOTE:** Node v14 also should work. However, that version is nearing [end-of-life](#). We recommend updating to Node 16 or 18.

Yarn (v1.x)

- Yarn v1.x is available at <https://classic.yarnpkg.com/>. It can usually be install via NPM (or through your Linux distribution's package manager). *We do NOT currently support Yarn v2.*

```
# You may need to run this command using "sudo" if you don't have proper privileges
npm install --global yarn
```

- Yarn is used to build the frontend.

PM2 (or another Process Manager for Node.js apps) (*optional, but recommended for Production*)

- In Production scenarios, we *highly recommend* starting/stopping the User Interface using a Node.js process manager. There are several available, but our current favorite is [PM2](#). The rest of this installation guide assumes you are using PM2.
- [PM2](#) is very easily installed via NPM


```
# You may need to run this command using "sudo" if you don't have proper privileges
npm install --global pm2
```

DSpace 7.x Backend (see above)

- The DSpace User Interface (Frontend) cannot function without an installed DSpace Backend. Follow the instructions above.
- The Frontend and Backend *do not need to be installed on the same machine/server*. They may be installed on separate machines as long as the two machines can connect to one another via HTTP or HTTPS.

Frontend Installation

Below instructions are specific to 7.2 (or later)

 The Frontend Instructions below are specific to 7.2 or later. For Frontend Installation instructions for 7.0 or 7.1, see [7.0-7.1 Frontend Installation](#)

1. **Download Code (to [dspace-angular]):** Download the [latest dspace-angular release](#) from the DSpace GitHub repository. You can choose to either download the zip or tar.gz file provided by GitHub, or you can use "git" to checkout the appropriate tag (e.g. dspace-7.2) or branch.
 - a. NOTE: For the rest of these instructions, we'll refer to the source code location as [dspace-angular].
2. **Install Dependencies:** Install all required local dependencies by running the following from within the unzipped [dspace-angular] directory

```
# change directory to our repo
cd [dspace-angular]

# install the local dependencies
yarn install


# NOTE: Some dependencies occasionally get overly strict over exact versions of Node & Yarn.
# If you are running a supported version of Node & Yarn, but see a message like
# `The engine "node" is incompatible with this module.` , you can disregard it using this flag:
# yarn install --ignore-engines
```

3. **Build/Compile:** Build the User Interface for Production. This builds source code (under [dspace-angular]/src/) to create a compiled version of the User Interface in the [dspace-angular]/dist folder. This /dist folder is what we will deploy & run to start the UI.

```
yarn build:prod
```

- a. You only need to rebuild the UI application if you change source code (under [dspace-angular]/src/). Simply changing the configurations (e.g. config.prod.yml, see below) do not require a rebuild, but only require restarting the UI.
4. **Deployment (to [dspace-ui-deploy]):** (*Only recommended for Production setups*) Choose/Create a directory on your server where you wish to run the compiled User Interface. We'll call this [dspace-ui-deploy].

[dspace-ui-deploy] vs [dspace-angular]

 [dspace-angular] is the directory where you've downloaded and built the UI source code (per the instructions above). For deployment /running the UI, we recommend creating an entirely separate [dspace-ui-deploy] directory. This keeps your running, production User Interface separate from your source code directory and also minimizes downtime when rebuilding your UI. You may even choose to deploy to a [dspace-ui-deploy] directory on a different server (and copy the /dist directory over via FTP or similar).

If you are installing the UI for the first time, or just want a simple setup, you can choose to have [dspace-ui-deploy] and [dspace-angular] be the *same directory*. This would mean you don't have to copy your /dist folder to another location. However, the downside is that your running site will become unresponsive whenever you do a re-build/re-compile (i.e. rerun "yarn build:prod") as this build process will first delete the [dspace-angular]/dist directory before rebuilding it.

- a. Copy the entire [dspace-angular]/dist/ folder to this location. For example:

```
cp -r [dspace-angular]/dist [dspace-ui-deploy]
```

- b. **WARNING:** At this time, you **MUST** copy the entire "dist" folder and make sure NOT to rename it. Therefore, the directory structure should look like this:

Contents of [dspace-ui-deploy] folder

```
[dspace-ui-deploy]
/dist
  /browser (compiled client-side code)
  /server (compiled server-side code, including "main.js")
/config (Optionally created in the "Configuration" step below)
  /config.prod.yml (Optionally created in the "Configuration" step below)
```

- c. NOTE: the OS account which runs the UI via Node.js (see below) MUST have write privileges to the [dspace-ui-deploy] directory (because on startup, the runtime configuration is written to [dspace-ui-deploy]/dist/browser/assets/config.json)
5. **Configuration:** You have two options for [User Interface Configuration](#), Environment Variables or YAML-based configuration (config.prod.yml). Choose one!

- a. **YAML configuration:** Create a "config.prod.yml" at [dspace-ui-deploy]/config/config.prod.yml. You may wish to use the [dSPACE-UI-DEPLOY]/config/config.example.yml as a starting point. This config.prod.yml file can be used to override any of the default configurations listed in the config.example.yml (in that same directory). *At a minimum* this file MUST include a "rest" section (and may also include a "ui" section), similar to the following (keep in mind, you only need to include settings that you need to modify).

Example config.prod.yml

```
# The "ui" section defines where you want Node.js to run/respond. It often is a *localhost* (non-
public) URL, especially if you are using a Proxy.
# In this example, we are setting up our UI to just use localhost, port 4000.
# This is a common setup for when you want to use Apache or Nginx to handle HTTPS and proxy
requests to Node on port 4000
ui:
  ssl: false
  host: localhost
  port: 4000
  nameSpace: /

# This example is valid if your Backend is publicly available at https://api.myspace.edu/server/
# The REST settings MUST correspond to the primary/public URL of the backend. Usually, this means
they must be kept in sync
# with the value of "dspace.server.url" in the backend's local.cfg
rest:
  ssl: true
  host: api.myspace.edu
  port: 443
  nameSpace: /server
```

- b. **Environment variables:** Every configuration in the UI may be specified via an Environment Variable. See [Configuration Override](#) in the [User Interface Configuration](#) documentation for more details. For example, the below environment variables provide the same setup as the config.prod.yml example above.

Example Environment Variables

```
# All environment variables MUST
# (1) be prefixed with "DSPACE_"
# (2) use underscores as separators (no dots allowed), and
# (3) use all uppercase

# "ui" section
DSPACE_UI_SSL = false
DSPACE_UI_HOST = localhost
DSPACE_UI_PORT = 4000
DSPACE_UI_NAMESPACE = /

# "rest" section
DSPACE_REST_SSL = true
DSPACE_REST_HOST = api.mydspace.edu
DSPACE_REST_PORT = 443
DSPACE_REST_NAMESPACE = /server
```

- i. NOTE: When using PM2, some may find it easier to use Environment variables, as it allows you to specify DSpace UI configs within your PM2 configuration. See PM2 instructions below.
- c. Configuration Hints:
 - i. See the [User Interface Configuration](#) documentation for a list of all available configurations.
 - ii. In the "ui" section above, you may wish to start with "ssl: false" and "port: 4000" just to be certain that everything else is working properly **before** adding HTTPS support. KEEP IN MIND, we highly recommend always using HTTPS for Production. (See section on HTTPS below)
 - iii. (Optionally) *Test the connection to your REST API from the UI from the command-line*. This is not required, but it can sometimes help you discover immediate configuration issues if the test fails.
 - 1. If you are using YAML configs, copy your config.prod.yml back into your source code folder at [dspace-angular] /config/config.prod.yml
 - 2. From [dspace-angular], run `yarn test:rest` This script will attempt a basic Node.js connection to the REST API that is configured in your "config.prod.yml" file and validate the response.
 - 3. A successful connection should return a 200 Response and all JSON validation checks should return "true"
 - 4. If you receive a connection error or different response code, you MUST fix your REST API before the UI will be able to work. See also the ["Common Installation Issues"](#) below. If you receive an SSL error, see ["Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend"](#)
 - iv. When using a subpath (nameSpace) in your UI server base URL (e.g. "<http://localhost:4000/mysite/>" instead of "<http://localhost:4000/>"), you must make sure that the URL **without** the subpath is added to the `rest.cors.allowed-origins` list in [dspace] /config/modules/rest.cfg or the `local.cfg` override. The default value used for this configuration assumes that Origin and DSpace URL are identical, but CORS origins do not contain a subpath.
- 6. **Start up the User Interface:** The compiled User Interface only requires [Node.js](#) to run. However, most users may want to use [PM2](#) (or a similar Node.js process manager) in Production to provide easier logging and restart tools.
 - a. *Quick Start:* To quickly startup / test the User Interface, you can just use Node.js. This is only recommended for quickly testing the UI is working, as no logs are available.

```
# You MUST start the UI from within the deployment directory
cd [dspace-ui-deploy]

# Run the "server/main.js" file to startup the User Interface
node ./dist/server/main.js

# Stop the UI by killing it via Ctrl+C
```

- b. *Run via PM2:* Using PM2 (or a different Node.js process manager) is highly recommended for Production scenarios. Here's an example of a Production setup of PM2.
 - i. First you need to create a PM2 JSON configuration file which will run the User Interface. This file can be named anything & placed where ever you like, but you may want to save it to your deployment directory (e.g. [dspace-ui-deploy] /dspace-ui.json).

dspace-ui.json

```
{
  "apps": [
    {
      "name": "dspace-ui",
      "cwd": "/full/path/to/dspace-ui-deploy",
      "script": "dist/server/main.js",
      "instances": "max",
      "exec_mode": "cluster",
      "env": {
        "NODE_ENV": "production"
      }
    }
  ]
}
```

1. NOTE: The "cwd" setting MUST correspond to your [dspace-ui-deploy] folder path.
2. NOTE #2: The "exec_mode" and "instances" settings are optional but highly recommended. Setting "exec_mode" to "cluster" enable's [PM2's cluster mode](#). This will provide better performance in production as it allows PM2 to scale your site across multiple CPUs. The "instances" setting tells PM2 how many CPUs to scale across ("max" means all CPUs, but you can also specify a number.)
3. NOTE #3: If you wanted to configure your UI using Environment Variables, specify those Environment Variables under the "env" section. For example:

Configuration via Environment Variables

```
"env": {
  "NODE_ENV": "production",
  "DSPACE_REST_SSL": "true",
  "DSPACE_REST_HOST": "demo.dspace.org",
  "DSPACE_REST_PORT": "443",
  "DSPACE_REST_NAMESPACE": "/server"
}
```

4. NOTE #4: If you are using Windows, there are two other rules to keep in mind in this JSON configuration. First, *all paths must include double backslashes* (e.g. "C:\\dspace-ui-deploy"). Second, "cluster" mode is *required*. Here's an example configuration for Windows:

dspace-ui.json (for Windows)

```
{
  "apps": [
    {
      "name": "dspace-ui",
      "cwd": "C:\\full\\path\\to\\dspace-ui-deploy",
      "script": "dist\\server\\main.js",
      "instances": "max",
      "exec_mode": "cluster",
      "env": {
        "NODE_ENV": "production"
      }
    }
  ]
}
```

- ii. Now, start the application using PM2 using the configuration file you created in the previous step


```
# In this example, we are assuming the config is named "dspace-ui.json"
pm2 start dspace-ui.json

# To see the logs, you'd run
# pm2 logs

# To stop it, you'd run
# pm2 stop dspace-ui.json

# If you need to change your PM2 configs, delete the old config and restart
# pm2 delete dspace-ui.json
```

- iii. For more PM2 commands see <https://pm2.keymetrics.io/docs/usage/quick-start/>
 - iv. HINT: You may also want to install/configure [pm2-logrotate](#) to ensure that PM2's log folder doesn't fill up over time.
 - v. *Did PM2 not work or throw an immediate error?* It's likely that something in your UI installation or configuration is incorrect. Check the PM2 logs ("pm2 logs") first for errors. If the problem is not obvious, try to see if you can run the UI using the "Quick Start" method (using just Node.js) instead. Once "Quick Start" is working, try PM2 again.
 - vi. *If neither PM2 nor the "Quick Start" method works for you:* then see the "User Interface never appears (no content appears)" section in the [Commons Installation Issues](#) below
7. **Test it out:** At this point, the User Interface should be available at the URL you configured!
- a. For an example of what the default frontend looks like, visit the Demo Frontend: <https://demo.dspace.org/>
 - b. If the UI fails to start or throws errors, it's likely a configuration issue. See [Commons Installation Issues](#) below for common error messages you may see and how to resolve them.
 - c. If you have an especially difficult issue to debug, you may wish to *stop* PM2. Instead, try running the UI via the "Quick Start" method (using just Node.js). This command might provide a more specific error message to you, if PM2 is not giving enough information back.
8. **Add HTTPS support:** For HTTPS (port 443) support, you have two options
- a. (*Recommended*) Install either [Apache HTTPD](#) or [Nginx](#) to act as a "reverse proxy" for the frontend (and backend). This allows you to manage HTTPS (SSL certificates) in either Apache HTTPD or Nginx, and proxy all requests to the frontend (running on port 4000) and backend (running on port 8080). This is our current recommended approach. These instructions are specific to Apache, but a similar setup can be achieved with Nginx.
 - i. If you already have Apache / Nginx installed for the backend, you can use the same Apache / Nginx. You can also choose to install a separate one (either approach is fine).
 - 1. Install [Apache HTTPD](#), e.g. `sudo apt install apache2`
 - 2. Install the [mod_proxy](#) and [mod_proxy_http](#) modules, e.g. `sudo a2enmod proxy; sudo a2enmod proxy_http`
 - 3. Restart Apache to enable
 - 4. Obtain an SSL certificate for HTTPS support. If you don't have one yet, you can use Let's Encrypt (for free) using the "certbot" tool: <https://certbot.eff.org/>
 - ii. *Apache HTTPD sample configuration:*
 - 1. Now, setup (or update) the new [VirtualHost](#) for your UI site (preferably using HTTPS / port 443) which proxies all requests to PM2 running on port 4000.

```

<VirtualHost _default_:443>
    # Add your domain here. We've added "my.dspace.edu" as an example
    ServerName my.dspace.edu
    .. setup your host how you want, including log settings...

    # Most installs will need these options enabled to ensure DSpace knows its
    hostname and scheme (http or https)
    # Also required to ensure correct sitemap URLs appear in /robots.txt for User
    Interface.
    ProxyPreserveHost On
    RequestHeader set X-Forwarded-Proto https

    # These SSL settings are identical to those for the backend installation (see
    above)
    # If you already have the backend running HTTPS, just add the new Proxy settings
    below.
    SSLEngine on
    SSLCertificateFile [full-path-to-PEM-cert]
    SSLCertificateKeyFile [full-path-to-cert-KEY]
    # LetsEncrypt certificates (and possibly others) may require a chain file be
    specified
    # in order for the UI / Node.js to validate the HTTPS connection.
    #SSLCertificateChainFile [full-path-to-chain-file]

    # These Proxy settings are for the backend. They are described in the backend
    installation (see above)
    # If you already have the backend running HTTPS, just append the new Proxy
    settings below.
    # Proxy all HTTPS requests to "/server" from Apache to Tomcat via AJP connector
    # (In this example: https://my.dspace.edu/server/ will display the REST API)
    ProxyPass /server ajp://localhost:8009/server
    ProxyPassReverse /server ajp://localhost:8009/server

    # [NEW FOR UI:] Proxy all HTTPS requests from Apache to PM2 on localhost, port
    4000
    # NOTE that this proxy URL must match the "ui" settings in your config.prod.yml
    # (In this example: https://my.dspace.edu/ will display the User Interface)
    ProxyPass / http://localhost:4000/
    ProxyPassReverse / http://localhost:4000/
</VirtualHost>

```

iii. NGinx sample configuration

1. Sample NGinx "server block" configuration. Keep in mind we are only providing basic example settings.

```
# Setup HTTPS access
server {
    listen 443 ssl;
    # Add your domain here. We've added "my.dspace.edu" as an example
    server_name my.dspace.edu;

    # Add your SSL certificate/key path here
    # NOTE: For LetsEncrypt, the certificate should be the full certificate chain file
    # These SSL settings are identical to those for the backend installation (see
    above)
    # If you already have the backend running HTTPS, just add the new Proxy settings
    below.
    ssl_certificate my.dspace.edu.crt (or PEM);
    ssl_certificate_key my.dspace.edu.key;

    # Proxy all HTTPS requests to "/server" from NGinx to Tomcat on port 8080
    # These Proxy settings are for the backend. They are described in the backend
    installation (see above)
    location /server {
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Host $host;
        proxy_pass http://localhost:8080/server;
    }

    # [NEW FOR UI:] Proxy all HTTPS requests from NGinx to PM2 on localhost, port 4000
    # NOTE that this proxy URL must match the "ui" settings in your config.prod.yml
    # (In this example: https://my.dspace.edu/ will display the User Interface)
    location / {
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Host $host;
        proxy_pass http://localhost:4000/;
    }
}
```

- iv. HINT#1: Because you are using a proxy for HTTPS support, in your [User Interface Configuration](#), your "ui" settings will still have "ssl: false" and "port: 4000". This is perfectly OK!
- v. HINT#2: to force the UI to connect to the backend using HTTPS, you should verify your "rest" settings in your [User Interface Configuration](#) **match** the "dspace.server.url" in your backend's "local.cfg" and both use the HTTPS URL. So, if your backend (REST API) is proxied to <https://my.dspace.edu/server/>, both those settings should specify that HTTPS URL.
- vi. HINT#3: to force the backend to recognize the HTTPS UI, make sure to update your "dspace.ui.url" in your backend's "local.cfg" is updated to use the new HTTPS UI URL (e.g. <https://my.dspace.edu>).
- b. (Alternatively) You can use the basic HTTPS support built into our UI and Node server. (This may currently be better for non-Production environments as it has not been well tested)
 - i. Create a [dspace-ui-deploy]/config/ssl/ folder and add a key.pem and cert.pem to that folder (they must have those exact names)
 - ii. In your [User Interface Configuration](#), go back and update the following:
 1. Set "ui > ssl" to true
 2. Update "ui > port" to be 443
 - a. In order to run Node/PM2 on port 443, you also will likely need to provide node with special permissions, like [in this example](#).
 - iii. Restart the UI
 - iv. Keep in mind, while this setup is simple, you may not have the same level of detailed, Production logs as you would with Apache HTTPD or Nginx

What Next?

After a successful installation, you may want to take a closer look at

- [Performance Tuning DSpace](#): If you are noticing any slowness in your Production site, we have a guide for how you might speed things up.
- [User Interface Customization](#): Documentation on customizing the User Interface with your own branding / theme(s)
- [User Interface Configuration](#): Additional configurations available in the User Interface.
- [Submission User Interface](#): Options to configure/customize the default Submission (deposit) process
- [Configurable Workflow](#): Options to configure/customize the default Workflow approval process
- [Scheduled Tasks via Cron](#) : Several DSpace features **require** that a command-line script is run regularly via cron.
- [Configuration Reference](#) : Details on the configuration options available to the Backend
- [Handle Server installation](#): Optionally, you may wish to enable persistent URLs for your DSpace site using CRNI's Handle.Net Registry
- [Statistics and Metrics](#): Optionally, you may wish to configuration one (or more) Statistics options within DSpace, including [Google Analytics](#) and (internal) [Solr Statistics](#)
- [Multilingual Support](#): Optionally, you may wish to enable multilingual support in your DSpace site.
- [Using DSpace](#) : Various other pages which describe usage and additional configurations related to other DSpace features.
- [System Administration](#): Various other pages which describe additional backend installation options/configurations.

If you've run into installation problems, you may want to...

- Visit the [Troubleshoot an error](#) guide for tips on locating the cause of the error
- Review [Commons Installation Issues](#) (see below)
- Ask for [Support](#) via one of the support options documented on that page

Common Installation Issues

Troubleshoot an error or find detailed error messages

See the [Troubleshoot an error](#) guide, look for the section on "DSpace 7.x". This will provide you hints on locating error messages both in the User Interface (frontend) and in the REST API (backend)

User Interface never appears (no content appears) or "Proxy server received an invalid response"

Chances are your User Interface (UI) is throwing a severe error or not starting properly. The best way to debug this issue would be to start the User Interface in development mode to see if it can give you a more descriptive error.

1. First, create a `[dspace-ui-deploy]/config/config.dev.yml` configuration file for development. This file supports the same configs as your existing `config.prod.yml`. So, you can copy over any settings you want to test out.
2. Start the UI in development mode (this doesn't require a proxy like Apache or Nginx)

```
yarn start:dev
```

3. This will boot up the User Interface on whatever port you specified in "config.dev.yml"
4. At this point, attempt to access the UI from your web browser. Even if it isn't fully working, you should be able to still get more information from your browser's DevTools regarding the underlying error. See the [Troubleshoot an error](#) page, look for the section on "DSpace 7.x". It has a guide for locating UI error messages in your browser's Developer Tools.

Once you've found the underlying error, it may be one of the "common installation issues" listed below.

User Interface partially load but then spins (never fully loads or some content doesn't load)

Chances are your User Interface (UI) is throwing an error or receiving an unexpected response from the REST API backend. Since the UI is Javascript based, it runs entirely in your browser. That means the error it's hitting is most easily viewed in your browser (and in fact the error may never appear in log files).

See the [Troubleshoot an error](#) page, look for the section on "DSpace 7.x". It has a guide for locating UI error messages in your browser's Developer Tools.

"500 Service Unavailable" from the User Interface

This error is saying that the frontend is working, but it is unable to communicate with your backend. It's the same as the "[No _links section found at...](#)" error described in the next section. Please follow the troubleshooting details in that section.

"No _links section found at..." error from User Interface

When starting up the User Interface for the first time, you may see an error that looks similar to this...

```
No _links section found at [rest-api-url]
ERROR Error: undefined doesn't contain the link sites
    at MapSubscriber.project
```

This error means that the UI is unable to contact the REST API listed at `[rest-api-url]` and/or the response from that `[rest-api-url]` is unexpected (as it doesn't contain the "`_links`" to the endpoints available at that REST API). A valid DSpace `[rest-api-url]` will respond with JSON similar to our demo API at <https://demo.dspace.org/server/api>

First, *test the connection to your REST API from the UI from the command-line.*

```
# This script will attempt a basic Node.js connection to the REST API
# configured in your "[dspace-angular]/config/config.prod.yml" and
# validate the response.(NOTE: config.prod.yml MUST be copied to
# to [dspace-angular]/config/ for this script to find it!)
yarn test:rest

# In DSpace 7.1 a different command was needed
# yarn config:check:rest
```

- A successful connection should return a 200 Response and all JSON validation checks should return "true".
- If you receive a connection error or different response code, you **MUST** fix your REST API before the UI will be able to work (see additional hints below for likely causes).

Usually, the core problem is caused by one of the following scenarios:

- *A possible configuration issue* in the frontend or backend.
 - Check the "rest" section of your `config.*.yaml` (or `environment.*.ts` for 7.1 or 7.0) configuration file for the User Interface. That configuration section defines which REST API the UI will attempt to use. If the settings do NOT map to a valid DSpace REST API, then you will see this "No _links section found.." error. Keep in mind, **the REST API must use HTTPS** (the only exception is if both the frontend and backend are running on "localhost"-based URLs)
 - Check the "dspace.ui.url" configuration of your backend & verify it corresponds to the public URL of the User Interface (i.e. the exact same URL you use in your browser)
 - Verify the backend "trusts" the frontend via the "rest.cors.allowed-origins" configuration (in `rest.cfg` or `local.cfg`). This setting must list all web-based clients which are trusted by the backend (REST API). By default, "dspace.ui.url" should be listed... but you should verify it has not been modified.
- *A possible SSL certificate issue*. This issue may also appear if the REST API's SSL Certificate is either *untrusted (by the frontend)* or *expired*.
 - If you are using a [Let's Encrypt](#) style certificate, you may need to modify your backend's Apache settings to also provide a Chain File as follows:

```
# For example: /etc/letsencrypt/live/[domain]/cert.pem
SSLCertificateFile [full-path-to-PEM-cert]
# For example: /etc/letsencrypt/live/[domain]/privkey.pem
SSLCertificateKeyFile [full-path-to-cert-KEY]
# For example: /etc/letsencrypt/live/[domain]/chain.pem
SSLCertificateChainFile [full-path-to-chain-file]
```

- Per the [Apache docs](#), you can also use the SSLCertificateFile setting to specify intermediate CA certificates along with the main cert.
- For self-signed certs, see also "[Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend](#)" common issue listed below.
- *Something blocking access to the REST API*. This may be a proxy issue, a firewall issue, or something else generally blocking the port (e.g. port 443 for SSL).
 - Verify that you can access the REST API from the machine where Node.js is running (i.e. your UI is running). For example try a simple "wget" or "curl" to verify the REST API is returning expected JSON similar to our demo API at <https://demo.dspace.org/server/api>

```
# Attempt to access the REST API via HTTPS from command-line on the machine where Node.js is
running.
# If this fails or throws a SSL cert error, you must fix it.
wget https://[rest.host]/server/api
```

- **In most production scenarios, your REST API should be publicly accessible on the web**, unless you are guaranteed that all your DSpace users will access the site behind a VPN or similar. So, this "No _links section found" error may also occur if you are accessing the UI from a client computer/web browser which is *unable to access the REST API*.

If none of the above suggestions helped, you may want to look closer at the request logs in your browser (using browser's Dev Tools) and server-side logs, to be sure that the requests from your UI are going where you expect, and see if they appear also on the backend. Tips for finding these logs can be found in the "DSpace 7.x" section of our [Troubleshoot an error](#) guide.

"RangeError: Maximum call stack size exceeded"

When starting up the User Interface for the first time, you may see an error that looks similar to this...

```
ERROR RangeError: Maximum call stack size exceeded
```

This error means that the UI is trying to contact your REST API, but is having issues doing so (possibly because either a proxy or an HTTPHTTPS redirect is causing issues or a redirect loop).

Double check your "dspace.server.url" setting in your `local.cfg` on the backend. Is it the same URL you use in your browser to access the backend? Keep in mind the mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash*.

Also double check the "rest" section of your `config.*.yaml` (or `environment.*.ts` for 7.1 or 7.0) configuration file for the User Interface. Make sure it's also pointing to the exact same URL as that "dspace.server.url" setting. Again, check the mode, domain, port and paths all match exactly.

"XMLHttpRequest.. has been blocked by CORS policy" or "CORS error" or "Invalid CORS request"

If you are seeing a CORS error in your browser, this means that you are accessing the REST API via an "untrusted" client application. To fix this error, you must change your REST API / Backend configuration to trust the application.

- By default, the DSpace REST API / Backend will only trust the application at `dspace.ui.url`. Therefore, you should first verify that your `dspace.ui.url` setting (in your `local.cfg`) exactly matches the *primary URL* of your User Interface (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match.

- If you need to trust *additional* client applications / URLs, those MUST be added to the `rest.cors.allowed-origins` configuration. See [REST API](#) for details on this configuration.
- Also, check your Tomcat (or servlet container) log files. If Tomcat throws a syntax or other major error, it may return an error response that triggers a CORS error. In this scenario, the CORS error is only a side effect of a larger error.

If you modify either of the above settings, you will need to restart Tomcat for the changes to take effect.

Cannot login from the User Interface with a password that I know is valid

If you cannot login via the user interface with a valid password, you should check to see what underlying error is being returned by the REST API. The easiest way to do this is by using your web browser's Dev Tools as described in our [Troubleshoot an error](#) guide (see the "Try this first" section for DSpace 7).

If the password is valid, more than likely you'll see the underlying error is **"403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token"** (see hints on solving this in the very next section)

"403 Forbidden" error with a message that says "Access is denied. Invalid CSRF Token"

First, double check that you are seeing that exact error message. A `403 Forbidden` error may be thrown in a variety of scenarios. For example, a 403 may be thrown if a page requires a login, if you have entered an invalid username or password, or even sometimes when there is a CORS error (see previous installation issue for how to solve that).

If you are seeing the message "Invalid CSRF Token" message (especially on every login), this is usually the result of a configuration / setup issue.

Here's some things you should double check:

1. If your site had been working, and this error seems random, it is possibly that `DSpace-XSRF-COOKIE` cookie in your browser just got "out of sync" (this can occur if you are logging into the REST API and UI separately in the same browser).
 - a. Logout and login & try the same action again. If it works this time, then that cookie was just "out of sync". If it fails a second time, then there is a likely configuration issue...see suggestions below.
2. **Make sure your backend is running HTTPS!** This is the most common cause of this error. The only scenario where you can run the backend in HTTP is when both the frontend & backend URLs are "localhost"-based URLs.
 - a. The reason for this HTTPS requirement is that most modern browsers will automatically block cross-domain cookies when using HTTP. Cross-domain cookies are *required* for successful authentication. The only *exception* is when both the frontend and backend are using localhost URLs (as in that scenario the cookies no longer need to be sent cross-domain). A more technical description of this behavior is in the sub-bullets below.
 - i. If the REST API Backend is running HTTP, then it will always send the required `DSpace-XSRF-COOKIE` cookie with a value of `SameSite=Lax`. This setting means that the cookie will *not* be sent (by your browser) to any other domains. Effectively, this will block all logins from any domain that is not the same as the REST API (as this cookie will not be sent back to the REST API as required for CSRF validation). In other words, running the REST API on HTTP is only possible if the User Interface is running on the exact same domain. For example, running both on 'localhost' with HTTP is a common development setup, and this will work fine.
 - ii. In order to allow for cross-domain logins, you MUST enable HTTPS on the REST API. This will result in the `DSpace-XSRF-COOKIE` cookie being set to `SameSite=None; Secure`. This setting means the cookie will be sent cross domain, but only for HTTPS requests. It also allows the user interface (or other client applications) to be on any domain, provided that the domain is trusted by CORS (see `rest.cors.allowed-origins` setting in [REST API](#))
3. Verify that your User Interface's "rest" section matches the value of `dspace.server.url` configuration on the Backend. This simply ensures your UI is sending requests to the correct REST API. Also pay close attention that both specify HTTPS when necessary (see previous bullet).
4. Verify that your `dspace.server.url` configuration on the Backend matches the primary URL of the REST API (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash* (e.g. "https://demo.dspace.org/server" is valid, but "https://demo.dspace.org/server/" may cause problems).
5. Verify that your `dspace.ui.url` configuration on the Backend matches the primary URL of your User Interface (i.e. the URL you see in the browser). This must be an exact match: mode (http vs https), domain, port, and subpath(s) all must match, and it *must not end in a trailing slash* (e.g. "https://demo.dspace.org" is valid, but "https://demo.dspace.org/" may cause problems).
6. Verify that nothing (e.g. a proxy) is blocking Cookies and HTTP Headers from being passed between the UI and REST API. DSpace's CSRF protection relies on the client (User Interface) being able to return both a valid `DSpace-XSRF-COOKIE` cookie and a matching `X-XSRF-TOKEN` header back to the REST API for validation. See our REST Contract for more details <https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md>
7. If you are running a custom application, or accessing the REST API from the command-line (or other third party tool like [Postman](#)), you MUST ensure you are sending the CSRF token on every modifying request. See our REST Contract for more details <https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md>

For additional information on how DSpace's CSRF Protection works, see our REST Contract at <https://github.com/DSpace/RestContract/blob/main/csrf-tokens.md>

Using a Self-Signed SSL Certificate causes the Frontend to not be able to access the Backend

If you setup the backend to use HTTPS with a self-signed SSL certificate, then Node.js (which the frontend runs on) may not "trust" that certificate by default. This will result in the Frontend not being able to make requests to the Backend.

One possible workaround (untested as of yet) is to try setting the `NODE_EXTRA_CA_CERTS` environment variable (which tells Node.js to trust additional CA certificates).

```
# May be necessary for self-signed certificates.
export NODE_EXTRA_CA_CERTS="/etc/ssl/my.dspace.pem"
```


Another option is to avoid using a self-signed SSL certificate. Instead, create a real, issued SSL certificate using something like [Let's Encrypt](#) (or similar free services)

My REST API is running under HTTPS, but some of its "link" URLs are switching to HTTP

This scenario may occur when you are running the REST API behind an HTTP proxy (e.g. Apache HTTPD's `mod_proxy_http`, Ngnix's `proxy_pass` or any other proxy that is forwarding from HTTPS to HTTP).

The fix is to ensure the DSpace REST API is sent the `X-Forwarded-Proto` header (by your proxying service), telling it that the forwarded protocol is HTTPS

```
X-Forwarded-Proto: https
```

In general, when running behind a proxy, the DSpace REST API depends on accurate `X-Forwarded-*` headers to be sent by that proxy.

My User Interface's robots.txt has incorrect sitemap URLs

This scenario may occur when you are running the User Interface behind an HTTP proxy (e.g. Apache HTTPD's `mod_proxy_http`, Ngnix's `proxy_pass` or any other proxy that is forwarding from HTTPS to HTTP).

The fix is to ensure the DSpace User Interface (frontend) is sent the correct `X-Forwarded-Proto` and `Host` (or `X-Forwarded-Host`) headers to tell it the correct hostname and scheme (HTTP or HTTPS)

Apache HTTPD example

```
ProxyPreserveHost on
RequestHeader set X-Forwarded-Proto https
```

Cannot upload file from User Interface

If everything seems to be working, but you cannot upload files, it's important to first check your logs for any possible backend errors. See the [Troubleshoot an error](#) page.

If you are running DSpace on a Debian-based system (e.g. Ubuntu), [some users have reported](#) that it's **required** grant "ReadWrite" access to Apache Tomcat (where the backend is running) via the service file (e.g. `/lib/systemd/system/tomcat9.service`). In the `[Service]` section you need to add something like this:

```
# Give Tomcat read/write on the DSpace installation
# Make sure to update the "/PATH/TO" to be the full path of your DSpace install
ReadWritePaths=/PATH/TO/dspace

# NOTE: If you don't want to give Tomcat read/write to all of DSpace,
# you could limit this further to just these folders
# dspace/assetstore
# dspace/solr
# dspace/log
```

Javascript heap out of memory

On some versions of Node.js or some operating systems, sites have reported seeing a "Javascript heap out of memory" error when trying to run the User Interface *in development mode* (`yarn start:dev`). This does not seem to occur on every system, but the fix is always the same. You should ensure that Node.js is given *at least* 4GB of memory via the "NODE_OPTIONS" environment variable

```
# Set the "NODE_OPTIONS" environment variable on your system. This example will work for Linux/macOS
# Ensure the "max-old-space-size" is set to 4GB (4096MB) or greater.
export NODE_OPTIONS=--max-old-space-size=4096
```

NOTE: More discussion on this issue can be found in <https://github.com/DSpace/dspace-angular/issues/2259> It appears to only occur on systems where the default memory allocated for Node isn't sufficient to build DSpace in development mode.

This same setting may also be used in production scenarios to give Node.js more memory to work with. See [Performance Tuning DSpace](#) for more details.

Solr responds with "Expected mime type application/octet-stream but got text/html" (404 Not Found)

This error occurs when Solr is either not initialized properly, or your DSpace backend is unable to find/communicate with Solr. Here's a few things you should double check:

1. Verify that Solr is running and/or check for errors in its logs. Try to restart it (usually via a command like `[solr]/bin/solr restart`), and verify it's accessible via `wget` or a web browser (usually at a URL like `http://localhost:8983/solr`)
2. Verify that your `solr.server` setting (in `local.cfg`) is correct for your Solr installation. This should correspond to the main URL of your Solr site (usually something like `http://localhost:8983/solr`). If you use `wget` or a browser from the machine running your DSpace backend, you should get a response from that URL (it should return the Solr Admin UI).
3. Verify that the required DSpace Solr cores have been properly installed/configured (per installation instructions above). When properly installed, you should be able to get a response from them. For example, the URL `${solr.server}/search/select` should run an empty query against the "search" core, returning an empty JSON result.
4. If Solr is running & you are sure `solr.server` is set properly, double check that nothing else could be blocking the DSpace backend from accessing Solr. For instance, if Solr is on a separate machine, verify that there is no firewall or proxy that could be blocking access between the DSpace backend machine and the Solr machine.

Database errors occur when you run `ant fresh_install`

There are two common errors that occur.

- If your error looks like this:

```
[java] 2004-03-25 15:17:07,730 INFO
      org.dspace.storage.rdbms.InitializeDatabase @ Initializing Database
[java] 2004-03-25 15:17:08,816 FATAL
      org.dspace.storage.rdbms.InitializeDatabase @ Caught exception:
[java] org.postgresql.util.PSQLException: Connection refused. Check
      that the hostname and port are correct and that the postmaster is
      accepting TCP/IP connections.
[java]      at
      org.postgresql.jdbc1.AbstractJdbc1Connection.openConnection(AbstractJd
bclConnection.java:204)
[java]      at org.postgresql.Driver.connect(Driver.java:139)
```

it usually means you haven't yet added the relevant configuration parameter to your PostgreSQL configuration (see above), or perhaps you haven't restarted PostgreSQL after making the change. Also, make sure that the `db.username` and `db.password` properties are correctly set in `[dspace]/config/dspace.cfg`. An easy way to check that your DB is working OK over TCP/IP is to try this on the command line:

```
psql -U dspace -W -h localhost
```

Enter the `dspace` database password, and you should be dropped into the `psql` tool with a `dspace=>` prompt.

- Another common error looks like this:

```
[java] 2004-03-25 16:37:16,757 INFO
      org.dspace.storage.rdbms.InitializeDatabase @ Initializing Database
[java] 2004-03-25 16:37:17,139 WARN
      org.dspace.storage.rdbms.DatabaseManager @ Exception initializing DB
      pool
[java] java.lang.ClassNotFoundException: org.postgresql.Driver
[java]      at java.net.URLClassLoader$1.run(URLClassLoader.java:198)
[java]      at java.security.AccessController.doPrivileged(Native
      Method)
[java]      at
      java.net.URLClassLoader.findClass(URLClassLoader.java:186)
```

This means that the PostgreSQL JDBC driver is not present in `[dspace]/lib`. See above.