# Install the Python Agent

**On this page:**



This topic describes how to prepare the application environment and install the AppDynamics Python Agent.

## Before You Begin

1. Verify support for your application environment at Python Supported Environments.
2. Verify that your Python environment meets the following requirements:
   - CPython Versions 2.6, 2.7, 3.4, 3.5, or 3.6
   - pip 1.4 or later
   - Access to the Python Package Index, https://pypi.python.org/ by the machine on which you will install the agent
3. Provide a WSGI-based application to monitor.
4. Verify that you can access the machine where the application runs as a user with privileges to install the agent software and restart the application. Verify that you have a user account with these privileges.
5. If you are using uWSGI, set `enable-threads=1` in the uWSGI configuration file. The agent requires multi-threading. There is a known incompatibility between the Python Agent and versions of uWSGI installed via OS packages managers such as 'apt-get'. For this reason, AppDynamics recommends installing uWSGI from pip to avoid this issue.
6. If the application to monitor runs in a virtual environment, activate the virtual environment. For example, the following source command activates a virtual environment:

   ```
   source /<path_to_virtual_environment>/bin/activate
   ```

   Activating a virtual environment is not necessary if the application runs in the global Python environment.

## Install the Agent

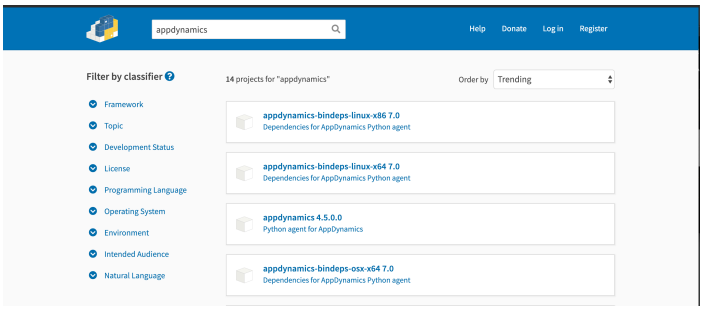
Log in to the machine on which the Python application runs using appropriate user credentials, as follows:

- For a virtual environment, you need to be the user who owns the virtual environment.
- For the global Python environment, you need to run the install command as root.

To install or upgrade to the latest version of the agent, run the `pip install` command as follows:

```
pip install -U appdynamics\<4.5
```

When there are multiple packages, you can locate the agent using the `pip list` command or using the List packages facility at https://pypi.python.org/pypi and then find `appdynamics` in the output. Here is a sample output:




For production deployments, AppDynamics recommends that you freeze the versions of dependencies so that they are not upgraded in production without first being deployed to your test/staging environments. The command to install or upgrade to a particular version of the AppDynamics Python Agent is:

```
pip install -U appdynamics==<released_agent_version>
```

For example:

```
pip install -U appdynamics==4.4.0.0
```

This command always installs the exact same version of the agent each time it is run.

## Configure the Agent

Provide a configuration file that specifies the required AppDynamics settings for agent-Controller communication. The file should be in Python ConfigParser format. The Python application user must have read access on the configuration file.

Note that lines in the configuration file must not start with spaces. Lines that begin with a # are comments and are ignored by the agent.

The following is a simple sample configuration file with minimum required settings.

```
[agent]
app = <app_name>
tier = <tier_name>
node = <node_name>

[controller]
host = <controller_host>
port = <controller_port>
ssl = on|off
account = <your AppDynamics controller account name>
accesskey = <your AppDynamics controller account access key>
```

Note the following points in the configuration:

1. The ssl settings determines whether the agent connects to the Controller by SSL. This is required for SaaS Controllers.
2. The `account` value is required if you are using a SaaS account or a multi-tenant on-premises controller. It defaults to `customer1` for a single tenant controller.
3. The accesskey is required for all controllers. To find your account name and access key, click the gear (  ) icon in the upper right corner of the AppDynamics UI, then click License.
4. Other settings, such as `ssl`, `http-proxy` or `wsgi_module`, may be required for your environment. See Python Agent Settings for a complete list of settings.

When you instrument an application using `pyagent run`, you pass the configuration file path as a parameter to the pyagent run comm and. In other deployments set the `APPD_CONFIG_FILE` environment variable as illustrated below in the samples for uWSGI with Emperor and Apache with mod_wsgi.

# Instrument the Application

Which instrumentation instructions to use depends on how the application is deployed, from these deployment options:

- pyagent run
- uWSGI with Emperor
- Apache with mod_wsgi

Irrespective of your Python environment, if you built your application using PasteDeploy, you can install the Python Agent by modifying the PasteDeploy configuration. See PasteDeploy.

## pyagent run

If you can control the way your WSGI server is invoked, you can instrument the application using `pyagent run`. This command runs your WSGI server with the Python agent enabled. This option is generally possible if you use a process launcher/manager that takes a command to execute. For example, frameworks managed by Supervisor, uWSGI without Emperor, init.d scripts, and so on.

To use the `pyagent run` command, prepend to your existing application run command the `pyagent run` command, passing the AppDynamics configuration file described in Configure the Agent as a parameter.

Do not overwrite `PYTHONPATH` for any reason. Doing so will prevent the `pyagent run` command from loading the agent. If you need to add to PYTHONPATH, use the pythonpath configuration variable. For example, these commands add `/foo` and `/bar` to the `PYTHONPATH` instead of overwriting it:

Correct way to add to PYTHONPATH:

```
pythonpath = /foo
pythonpath = /bar
```

Do not add the values to PYTHONPATH using the follwoing syntax. This is the wrong way to add to PYTHONPATH:

```
env = PYTHONPATH=/foo:/bar
```

### Using supervisorctl

If you use `supervisorctl`, after updating your Supervisor configuration you must use the `supervisorctl reload` command to have the Python agent loaded. Supervisor does not re-read its configuration files when you use the `supervisorctl restart` comm and.

To verify that the agent was loaded, look for the Python agent log file. Its default location is `/tmp/appd/logs/<app_name>-<node_name>.log`. For example, if your application name is `myapp` and your node name is `mynode` as specified in the agent configuration file, and you have not changed the location of the log file, the log file will be `/tmp/appd/logs/myapp-mynode`.

If the log file exists, the agent was loaded. If the log file does not exist, the agent was not loaded, in which case you should try reloading the Supervisor configuration with `supervisorctl reload`.

### Django and Flask

If your framework is Django or Flask, simply prepend `pyagent run` to your run command. For example, if your current run command looks like this:

```
gunicorn -w 8 -b '0.0.0.0:9000' example.app:application
```

Replace it with the following:

```
pyagent run -c <path_to_appdynamics_config_file> -- gunicorn -w 8 -b
'0.0.0.0:9000' example.app:application
```

### Other Pure Python WSGI-Based Frameworks

If you use a WSGI-based framework that is not Django or Flask:

1. In the AppDynamics configuration file, specify your WSGI application by setting the `APPD_WSGI_MODULE` directive to point to your app module. See Python Agent Settings.
2. Prepend `pyagent run` to your run command.
3. Run the AppDynamics-generated application.

For example, if your run command looks like this:

```
gunicorn -w 8 -b '0.0.0.0:9000' example.app:application
```

Replace it with these two commands:

```
pyagent run -c /path/to/appdynamics.cfg -- gunicorn -w 8 -b
'0.0.0.0:9000'
appdynamics.scripts.wsgi:application
```

## uWSGI with Emperor

If your environment is uWSGI with Emperor, you need to modify your WSGI configuration files and then manually launch the proxy.

uWSGI Emperor is a process manager specific to the uWSGI server. It does not allow you to control how the uWSGI processes that it manages are launched and therefore cannot be used with the `pyagent run` command.

The location of the WSGI configuration files is deployment-dependent. See http://uwsgi-docs.readthedocs.org/en/latest/Emperor.html for details of Emperor deployment.

### To instrument an application for uWSGI with Emperor:

1. Create the configuration file described in Configure the Agent.
2. Modify the uWSGI configuration file. Do one of the following, depending on whether the configuration uses a module directive or a wsgi-file directive:
   **Module Directive**
   If the uWSGI configuration has a module directive such as the following:

   ```
   module = yourcompany.sample:app
   ```

   modify that configuration by changing the module setting and adding the `APPD_WSGI_MODULE` and `APPD_CONFIG_FILE` settings to look like this, assuming that you have stored the configuration file in `/etc/appdynamics.cfg`:

   ```
   env = APPD_CONFIG_FILE=/etc/appdynamics.cfg
   env = APPD_WSGI_MODULE=yourcompany.sample:app
   module = appdynamics.scripts.wsgi:application
   ```

**WSGI-File Directive**
If the uWSGI configuration has a `wsgi-file` directive:

```
wsgi-file = /var/www/yourcompany/sample.py
callable = app
```

Modify the configuration to look like the following, assuming you have stored the configuration file in `/etc/appdynamics.cfg`:

```
env = APPD_CONFIG_FILE=/etc/appdynamics.cfg
env = APPD_WSGI_SCRIPT_ALIAS=/var/www/yourcompany/sample.py
env = APPD_WSGI_CALLABLE_OBJECT=app
module = appdynamics.scripts.wsgi
```

3. Before running any traffic through the instrumented application, manually launch the proxy by executing:

```
pyagent proxy start
```

## Apache with mod_wsgi

The Python Agent beta supports only `mod_wsgi` configurations that use `WSGIScriptAlias` that point to a single WSGI file. For example, the following type of configuration is supported:

```
WSGIScriptAlias /books /var/www/acme/bookstore/app.wsgi
```

If, instead, the script alias points to a directory, or if the script is using the `WSGIScriptAliasMatch` directive, contact python@appdynamics.com to discuss how the Python Agent can be deployed in your environment.

If the environment is Apache with `mod_wsgi` with a supported configuration as described above, you need to modify its `mod_wsgi` configuration files and manually launch the proxy.

To instrument an app for Apache with mod_wsgi:

1. Create the configuration file described in Configure the Agent.
2. Modify the `mod_wsgi` configuration file.
   If the `mod_wsgi` configuration file has an entry like this:

```
WSGIScriptAlias /books /var/www/acme/bookstore/app.wsgi
WSGICallableObject application
```

modify it to look like this, assuming that you have stored the configuration file in `/etc/appdynamics.cfg`:

```
SetEnv APPD_CONFIG_FILE /etc/appdynamics.cfg
SetEnv APPD_WSGI_MODULE acme.bookstore:app
WSGIScriptAlias /books
/<path_to_virtualenv>/lib/python2.7/site-packages/appdynamics/s
cripts/wsgi.py
```

3. Before running any traffic through the instrumented app, manually launch the proxy by executing:

```
pyagent proxy start
```

## PasteDeploy

You can instrument a Python application built with PasteDeploy by modifying your PasteDeploy configuration to use a composite factory supplied by AppDynamics. This feature can be used to instrument applications described by the other deployment options if they were built with PasteDeploy.

The AppDynamics composite factory is named `egg:appdynamics#instrument`. It requires a parameter named `target` that points to the application to the original application and the full path to the APPD_settings.

To instrument an application built with PasteDeploy:

1. Manually launch the AppDynamics proxy:

```
pyagent proxy start
```

2. In the PasteDeploy configuration file, rename the existing composite to a unique name.
   For example, if the existing composite configuration for an application named `metadata` is:

```
[composite:metadata]
use = egg:Paste#urlmap
/: meta
```

you could rename it:

```
[composite:_orig_metadata]
use = egg:Paste#urlmap
/: meta
```

3. Create a new composite section for the `metadata` application above the original one that you just renamed, as follows:
    a. Give the name of the old renamed application to the new composite application.
    b. Configure it to use the AppDynamics composite factory: `egg:appdynamics#instrument`.
    c. Set its target to the renamed application.
    d. Set the AppDynamics configuration file environment variable, `APPD_CONFIG_FILE`, to the path of your configuration file. For example:

```
[composite:metadata]
use = egg:appdynamics#instrument
target = orig_metadata
APPD_CONFIG_FILE = /etc/appdynamics.cfg
```

    You can also set other APPD_ configuration variables here. For example, `APPD_LOGS_DIR=/var/log/appdynamics`.
4. Restart the application.