Progress®



Run PAS for OpenEdge in a Docker Container

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

1

Overview

PAS for OpenEdge is available as a container so developers can develop, deploy, and run applications in an isolated container environment. This book describes how to configure PAS for OpenEdge containers.

For details, see the following topics:

- Purpose
- Audience
- What is Docker?
- Images and Containers
- Why use PAS for OpenEdge with Docker Containers?
- References

Purpose

This manual describes how to use Docker container technology with Progress Application Server for OpenEdge (PAS for OpenEdge). It describes how to create Docker container for PAS for OpenEdge and how to configure additional functionality once a container is set up.

Audience

This manual is intended for ABL developers and OpenEdge system administrators who are familiar with PAS for OpenEdge. Familiarity with Docker technology is beneficial, but not required to understand this manual. Additional learning sources are referenced throughout this manual.

What is Docker?

Docker is a technology that allows developers to develop, deploy, and run an application in a standalone, virtualized environment called a container. Containers are a lightweight, standardized, and secure method of executing software in an isolated environment. Using containers to isolate applications allows for faster and easier development and deployment.

Virtual machines offer a similar benefit to containers, but containers have the advantage because they do not virtualize hardware. Containers work on the application layer, and multiple containers share the same operating system kernel. This technology makes running multiple containers much more portable and efficient than running multiple virtual machines.

Continuous integration and continuous deployment (CI/CD) is one of the biggest benefits of using Docker containers. CI/CD allows developers to build, test, and deploy code in a wide variety of environments and allows them to test as frequently as they desire. By combining the development and testing processes, CI/CD and Docker containers can make developing, testing, and deploying software much more efficient.

For more information on Docker, see What is a Container and the Docker Documentation

Images and Containers

A Docker image is the core component of a container. Docker containers build images by reading instructions from a Dockerfile. A Docker container image has everything needed by the Docker daemon to instantiate the container and run an application. A Docker container image can be used to run multiple containers, allowing for flexibility and scalability.

A container is a runnable instance of a Docker image. A Docker container is defined by a Docker image, it includes its own execution environment with the code, runtime, system tools, libraries, and settings needed to run.

Why use PAS for OpenEdge with Docker Containers?

Docker containers provide several advantages when used with PAS for OpenEdge:

- Using Docker containers helps reduce overhead for packaging and deploying ABL applications, allowing you to streamline your application distribution processes. You only need to create a Docker custom container image with PAS for OpenEdge once, then you can distribute that image and run as many containers as needed.
- Similarly, when upgrading your Docker containers for PAS for OpenEdge, you only need to upgrade one image, and then you can easily redistribute the upgraded image.
- Clustering tools such as Docker Swarm and Kubernetes allow you to run multiple PAS for OpenEdge containers in a cluster, which provides high availability and elasticity.

References

Use the following resources to better understand how to use Docker with PAS for OpenEdge:

- PAS for OpenEdge Documentation
- Docker Documentation
- Kubernetes Documentation
- PAS for OpenEdge Sample Container

2

Build an Image and Run a Container

The following process allows you to build and deploy an OpenEdge application to a PAS for OpenEdge server using Docker container technology.

For details, see the following topics:

- Main Steps
- Docker Deployment Workflow

Main Steps

This workflow is done by first creating a custom Docker container image. You need to build an image for your application that combines the PAS for OpenEdge service artifacts and application files. With your custom container image, you can then create containers of your application:

1. The first step in the process is to export a packaged version of your PAS for OpenEdge application. For more information on that, see the PAS for OpenEdge Documentation.
2. The next step is to create a Dockerfile, which contains the instructions to create your custom container image. See the Create a Dockerfile section or details.

   Note: After creating an image it should be checked into a Docker repository.

3. Once you have your custom container image, you can then create and run a Docker container. You can perform this step in the following ways:

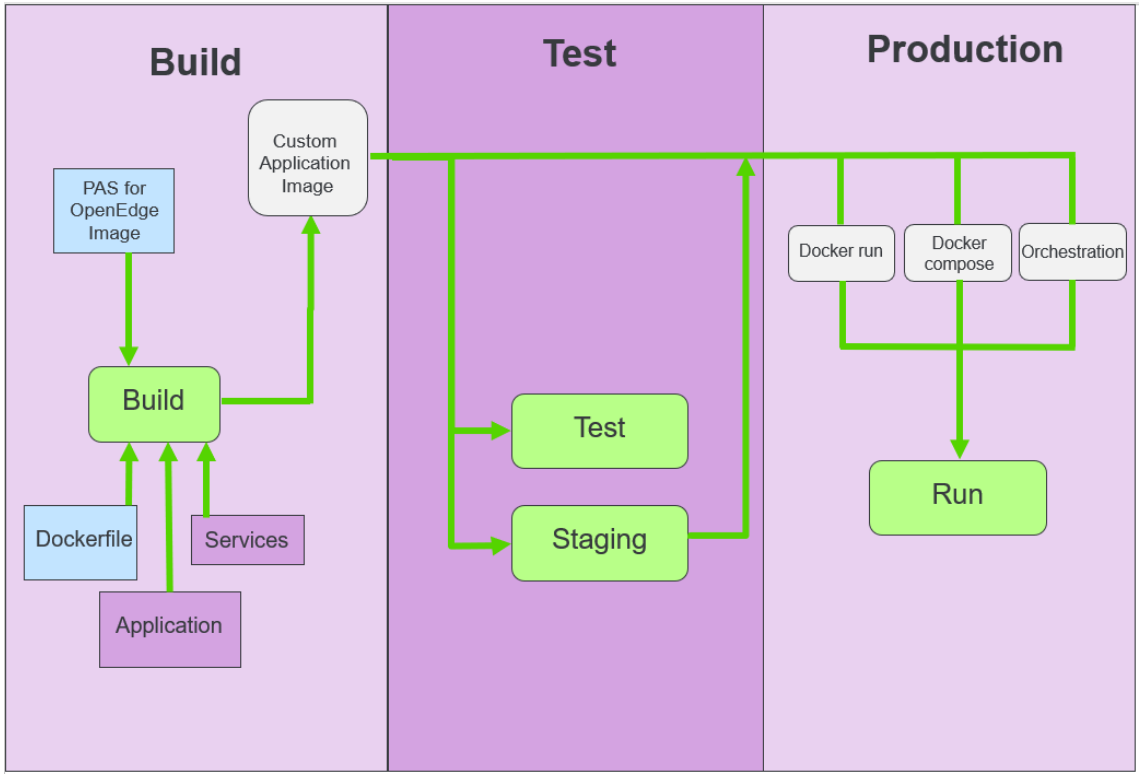
   - The docker run command
   - Docker Compose
   - Orchestration software such as Kubernetes or Docker Swarm

The docker run command and Docker Compose processes are described in the Run the Docker Container using the docker run Command and Run the Docker Container using Docker Compose sections. Orchestration software will not be covered in this document, for more information on that see the Kubernetes Documentation and the Docker swarm documentation.

Note: This workflow can also be implemented with industry tools designed for CI/CD.

Docker Deployment Workflow

The following diagram illustrates the high level workflow for using Docker with PAS for OpenEdge:



1. Build your custom application. This image is created using the PAS for OpenEdge image, a completed Dockerfile, your PAS for OpenEdge application, and any configured services you require.
2. Test your custom application image. Check for any issues and ensure that it is working properly.
3. After testing you can optionally stage your custom application image for further testing.
4. Enter Production using `docker run`, `docker compose`, or orchestration.
5. Run your custom application image.

3

Create a Dockerfile

To create a custom container image, you first need to create a Dockerfile. A Dockerfile contains all the commands necessary to create an image.

To create a Dockerfile:

Note: The code examples in these steps are taken from the ‘pasoe-basic’ sample app.

1. Specify the base image for PAS for OpenEdge in your Dockerfile. The custom container image is created from this base image. All subsequent commands modify this image. For example, the sample Dockerfile contains this code:

   ```
   FROM progress/pasoe
   ```

2. Define any environment variables:

   ```
   ENV INSTANCE_NAME oepas1
   ENV DLC /psc/dlc
   WORKDIR /psc/wrk
   ```

3. Add the command to create a PAS for OpenEdge instance:

   ```
   RUN /psc/dlc/bin/pasman create -v ${INSTANCE_NAME}
   ```

4. Copy any necessary application files:

   ```
   COPY --chown=pscadmin conf/sports.props service/Sports.war /psc/wrk/
   COPY --chown=pscadmin conf/logging.xml /psc/wrk/${INSTANCE_NAME}/conf/
   ```

5. Copy any necessary ABL files:

```
ADD service/openedge.tar /psc/wrk/${INSTANCE_NAME}/openedge

USER root
RUN chown -R pscadmin:pscadmin /psc/wrk/${INSTANCE_NAME}/openedge
```

6. Create a link to the logs folder:

```
RUN ln -s /psc/wrk/${INSTANCE_NAME}/logs /oelogs
```

7. Update the startup parameters:

```
RUN /psc/wrk/${INSTANCE_NAME}/bin/oeprop.sh -v -f /psc/wrk/sports.props
```

8. Deploy the REST service for your Dockerfile:

```
RUN /psc/wrk/${INSTANCE_NAME}/bin/tcman.sh deploy -v /psc/wrk/Sports.war

EXPOSE 8811
```

9. Start the PAS for OpenEdge instance:

```
COPY--chown=pscadmin scripts/start-pasinstance.sh /
RUN chmod +x /start-pasinstance.sh

CMD ["/start-pasinstance.sh"]
```

The `start-pasinstance.sh` file contains the following code:

```
#!/bin/bash

set -e
export DLC=/psc/dlc
export PATH=$DLC/bin:$PATH

# Start PASOE instance
cd /psc/wrk
/psc/wrk/${INSTANCE_NAME}/bin/tcman.sh start

# SIGTERM is issued by Docker commands (docker stop or docker-compose down)
# Adding a trap for the SIGTERM signal. i.e., to stop PASOE before stopping container
# not_done is a temporary flag
trap 'rm /tmp/not_done; /psc/wrk/${INSTANCE_NAME}/bin/tcman.sh stop; sleep 10' SIGTERM

wait_for_sigterm() {
  # Loop until not_done flag is removed
  touch /tmp/not_done
  while [ -f /tmp/not_done ]
  do
      sleep 60 &
      wait $!  # This is required such that a trap on main process is triggered
  done
}
if [ "$EFK_LOGGING" = "true" ]
then
    /usr/sbin/crond &
    /usr/local/bin/fluent-bit -c /etc/fluent-bit/fluent-bit.conf &
    wait_for_sigterm
else
   wait_for_sigterm
fi
```

The following is an example of a complete Dockerfile:

```
FROM progress/pasoe

ENV INSTANCE_NAME oepas1
ENV DLC /psc/dlc
WORKDIR /psc/wrk

# Create PASOE instance
RUN /psc/dlc/bin/pasman create -v ${INSTANCE_NAME}

# Copy application files - startup parameters - REST service
COPY --chown=pscadmin conf/sports.props service/Sports.war /psc/wrk/
COPY --chown=pscadmin conf/logging.xml /psc/wrk/${INSTANCE_NAME}/conf/

# Copy ABL files - extracting from tar file
ADD service/openedge.tar /psc/wrk/${INSTANCE_NAME}/openedge

USER root
RUN chown -R pscadmin:pscadmin /psc/wrk/${INSTANCE_NAME}/openedge

# Create symbolic link to logs folder
RUN ln -s /psc/wrk/${INSTANCE_NAME}/logs /oelogs

#Install cron support
RUN yum -y install cronie &&\
        yum clean all &&\
        rm -rf /var/cache/yum/

COPY scripts/remove-old-logs /etc/cron.d/remove-old-logs

RUN chmod 0644 /etc/cron.d/remove-old-logs &&\
        chmod u+s /usr/sbin/crond &&\
        touch /var/log/cron.log

USER pscadmin

RUN ls -l /psc/wrk &&\
        ls -l /psc/wrk/${INSTANCE_NAME}/openedge

# Update startup parameters
RUN /psc/wrk/${INSTANCE_NAME}/bin/oeprop.sh -v -f /psc/wrk/sports.props

# Deploy REST service
RUN /psc/wrk/${INSTANCE_NAME}/bin/tcman.sh deploy -v /psc/wrk/Sports.war

EXPOSE 8811

# CMD /bin/bash
COPY --chown=pscadmin scripts/start-pasinstance.sh /
RUN chmod +x /start-pasinstance.sh

CMD ["/start-pasinstance.sh"]
```

For details, see the following topics:

- Use the docker run Command
- Use Docker Compose

Use the docker run Command

Use the `docker run` command to instantiate and run a Docker container. The command creates a container from the specified image and then runs that container. Before using the `docker run` command, you can use the `docker build` command to create an image.

The following example builds an image from the Dockerfile in the current directory and tags that image:

```
docker build -t my_cust_app .
```

The following example creates and runs that Docker container:

```
docker run -dit -name my_cust_app1 <additional options> my_cust_app
```

The following example creates and runs a Docker container and sets several configuration options:

```
docker run -dit --name app1_sports_1 -e DBSERVER=192.168.56.1 -p 8811:8811 \
    -v `pwd`/conf/progress.cfg.pasoe:/psc/dlc/progress.cfg \
    -v `pwd`/logs:/psc/wrk/oepas1/logs \
    -v `pwd`/temp:/psc/wrk/oepas1/temp \
    --group-add 1000 app1_sports
```

Depending on your environment you can use different configuration options such as `-v` (volumes) and `-e` (environment variables).

See also:

- command-line entry for docker run
- reference material for docker run

Use Docker Compose

Docker compose is a tool for defining and running complex, multi-container Docker applications. A Docker compose file is a YAML (`docker-compose.yaml`) file, which contains configuration details of one or more services that represent the different parts of your application. For example, a service can represent your business logic while another represents the client that will access the business logic.

To run Docker containers using Docker compose use the following command:

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

See also:

- Docker compose documentation

In the following example `docker-compose.yaml` contains configuration details of the `web` and `sports` services:

- The web service represents the client access
- The sports service exposes the backend Data Service

```
version: '2'
services:
  web:
    image: nginx
    ports:
    - "8080:80"
    volumes:
    - ./webui:/usr/share/nginx/html:ro
  sports:
    build: .
    group_add:
    - 1000
    environment:
    - DBSERVER= 10.0.75.1
    ports:
    - "8811:8811"
    stdin_open: true
    tty: true
    volumes:
    - ./conf/progress.cfg.pasoe:/psc/dlc/progress.cfg
    - ./conf/sports.props:/psc/wrk/oepas1/logs
```

Note:

- This example uses version 2 of `docker-compose` because the `group_add` property is not included in version 3 . To run this example, change the `group_add` option to use your `group id` to avoid permission issues when reading and writing to files and folders mapped via the volumes option.
- The `sports` service uses the `build` property to first build the image and then run a container based on that image.
- The `web` service uses the `image` property to run a container based upon the specified image.
- The `docker-compose.yaml` file can be constructed differently. For example, you could have separate build and deploy phases and instead of building a service in the `docker-compose.yaml` file you could reference existing services created in an earlier deployment phase.

See also:

- Docker compose documentation

4

License a PAS for OpenEdge Docker Container

By default, PAS for OpenEdge Docker container image contains an empty license file. To integrate license support for your custom container image, replace the empty file with a proper license.

You can use the `volume` functionality of the PAS for OpenEdge Docker container to replace the license.

You can use Docker volumes in the following ways:

- In the `docker-compose` file
- By executing the docker `-v` command-line option
- Using Kubernetes

Perform the following steps to integrate the PAS for OpenEdge license using the `docker-compose` file:

1. Generate the license file for PAS for OpenEdge using the Customer Self Service (CSS) portal. The license file is a `progress.cfg` file.
2. Place the `progress.cfg` file in a directory on the host computer.
3. In the `docker-compose` file created earlier, in the section in which the PAS for OpenEdge service is defined, declare a mapping between the host directory of the license file and the directory in which PAS for OpenEdge is installed. `The progress.cfg` file is in the `/psc/dlc/` folder in the PAS for OpenEdge image.

The following example illustrates how the Docker volumes are used in a `docker-compose` file to integrate the `progress.cfg` file into a PAS for OpenEdge image:

```
sports:
  build: .
  environment:
```

```
- DBSERVER=192.168.99.1
ports:
- "8811:8811"
stdin_open: true
tty: true
volumes:
- /home/testuser/apps/myapp/conf/progress.cfg.pasoe:/psc/dlc/progress.cfg
```

See also:

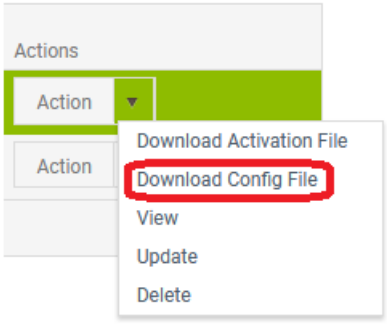
- Docker Documentation

For details, see the following topics:

- Generate a progress.cfg file

Generate a progress.cfg file

1. Log into the CSS portal with Progress account credentials.
2. Search for your account using Customer Identification details (either Customer Number or Name).
3. Navigate to the Account Info page.
4. Select Account -> Licenses -> Download OpenEdge Activation Files.
5. Download the `progress.cfg` file for PAS for OpenEdge:



5

Manage PAS for OpenEdge Docker Containers

When you run PAS for OpenEdge in a Docker container, log files are generated in the `logs` directory for the instance in the container (just as log files are generated on a host computer). But the files are lost when a container is removed or deleted. To preserve the contents of the logs directory, use Docker volumes to store and access your log files. The volumes functionality provides the ability to share data between the host file system and the Docker container.

A temp directory, (referenced by `-T` in the `openedge.properties agentStartupParam`) is created when you run PAS for OpenEdge. Unlike the log files, these temporary files do not need to be preserved but for performance reasons, Docker recommends that you store files created inside your container on the host machine. The same host directory can be mapped to multiple PAS for OpenEdge containers without loss of data.

To use the volume functionality, add a command to mount a directory on the host computer as a volume to the appropriate Docker container. You can mount a volume using the `docker-compose` file and the `docker run` command-line parameter, `--volume` or `-v`.

See also

- Docker storage overview

For details, see the following topics:

- Use a docker-compose file
- Use docker run -v

Use a docker-compose file

Use a `docker-compose` file if you are running a multi-container PAS for OpenEdge application. In the docker-compose file, configure a volume for every container whose data you want to preserve. Volumes accepts as values mapping between the directory on your host computer and the container directory in which the files for the instance reside.

The following example shows how to declare a volume mount in a `docker-compose` file:

```
version: '2'
services:
  sports:
    build: .
    ports:
    - "8810:8810"
    stdin_open: true
    tty: true
    volumes:
    - /home/testuser/apps/myapp-logs:/psc/wrk/oepas1/logs
    - /home/testuser/apps/myapp-temp:/psc/wrk/oepas1/temp
```

In the example, `/home/testuser/apps/myapp-logs` and `/home/testuser/apps/myapp-temp` are directories on the host computer and `/psc/wrk/oepas1/logs` and `/psc/wrk/oepas1/temp` are directories in the container.

Use docker run -v

If your business requires you to run a single-container application, use the parameter `-v` or `--volume`, with the `docker run` command to mount a host directory as a volume to a container directory.

The following command mounts the `/home/testuser/apps/myapp-logs` directory on the host computer as a volume to the `/psc/wrk/oepas1/logs` directory in the container. It does the same for the `/psc/wrk/oepas1/temp` directory.

```
docker run -it --name myapp
-v /home/testuser/apps/myapp-logs:psc/wrk/oepas1/logs
-v /home/testuser/apps/myapp-temp:psc/wrk/oepas1/temp   progress/pasoe
```

In the example, the `--name` parameter is used to specify the name of the container.

Note: When creating containers with the base PAS for OpenEdge image, you need to specify the `-i` (`--interactive`) option for the Docker command. Use `stdin_open: true` when working with Docker compose. If required you can also specify the `-t` (`--tty`) option, which is `-tty: true` in Docker compose.

6

Customize a PAS for OpenEdge Docker Container

Docker can build a custom container image by reading the instructions from a Dockerfile. This topic provides examples of how to update a Dockerfile to customize PAS for OpenEdge within a custom container image.

Note: The base image is a lightweight version of PAS for OpenEdge. Files specific to certain components, including Classic WebSpeed and ADM2, have been removed. Add the missing files by copying them from a corresponding installation, if required.

A Dockerfile contains multiple instructions to create a custom container image based on an existing image (`progress/pasoe`). For example, the Dockerfile can have an instruction to create a PAS for OpenEdge instance and deploy business logic to it. The created image has the same settings as that of the PAS for OpenEdge Docker container. Since HTTP is disabled in the base image, the newly created container has HTTP disabled by default, as will any other container created from that custom container image.

You can configure PAS for OpenEdge in your custom container image by adding command-line instructions in the Dockerfile.

To create and run an instance in a Docker container with custom settings, perform the following steps:

1. Create a `.props` file in a host directory. This file contains the desired custom settings for the instance.
2. Edit the Dockerfile to add an instruction to copy the `.props` file from the host directory to the container directory, `/psc/wrk`.

```
COPY /home/testuser/conf/pasoe_instance.props /psc/wrk
```

3. If you are also updating the `openedge.properties` file, add an additional instruction in the Dockerfile to update the properties file in the `psc/wrk/instance_name/conf` directory with the settings in the `/psc/wrk/.props` file.

```
RUN export DLC=/psc/dlc &&\
 cd /psc/wrk &&\
 /psc/wrk/oepas1/bin/oeprop.sh -v -f /psc/wrk/pasoe_instance.props
```

Note: The `-v` parameter is optional. Use the parameter if you want to see execution details for the command. The `-f` parameter requires the name of the file (`pasoe_instance.props`) whose contents are to be merged with the `openedge.properties` file of the instance.

4. Save and close the Dockerfile.
5. Build the custom container image using the modified Dockerfile.
6. Create and run a container from the updated image.

For details, see the following topics:

- Enable the APSV Adapter
- Update the Number of Agents in your PAS for OpenEdge instance
- Configure Environment Variables

Enable the APSV Adapter

To enable an APSV adapter in an instance, add the following settings to your `host .props` file:

```
# pasoe_instance.props file
[pasoe_instance.ROOT.APSV]
adapterEnabled=1
```

Update the Number of Agents in your PAS for OpenEdge instance

To set the number of agents that a session manager can manage, add the following settings to your `host .props` file:

```
[AppServer.SessMgr]
numInitialAgents=2
maxAgents=20
```

Configure Environment Variables

Docker offers various ways for setting an environment variable for a container. To set environment variables using a `docker-compose` file, use the environment key:

```
environment:
    - DBSERVER=10.128.57.41
    - MY_ENV=/test/user1/bin
```

To set environment variables using the `docker run` command, use the `--env-file` parameter with `env.list`. For example:

```
docker run --env-file env.list
```

Where `env.list` contains one or more environment variables.

For example, `env.list` can contain both the variables previously set using the `docker-compose` file. See Setting Environment Variables Using docker run for more information.

Note: Another option to customize your configuration is to create a file with the desired property settings, and then copy that file into the corresponding container directory.

See also:

- Setting Environment Variables using docker-compose

7

Upgrade a PAS for OpenEdge image in a container

You may need to update your PAS for OpenEdge container to a newer version of PAS for OpenEdge. The new image may be based on a newer version of CentOS, a new service pack release, or a new PAS for OpenEdge release. Do not modify a container to upgrade the OS or the PAS for OpenEdge server. Docker containers are immutable—any changes made to a container will be lost once the container is removed. Instead, rebuild the image based on the new version of PAS for OpenEdge. All newly created containers will be based on the new base image.

To upgrade a custom container image:

1. Download the new base PAS for OpenEdge image from the specified location.
2. Load the image into your Docker environment.
3. Integrate license support.
4. Build a custom container image based on the image loaded into the Docker environment. Use a Dockerfile, and a `docker-compose` file (if required), to build the custom container image.

   You can use the `docker tag` functionality to tag a custom container image. The tag functionality enables easy access to a specific build or version of an application.
5. Test the custom container image.
6. If the test succeeds as part of your CI/CD pipeline, use the image in your production environment. To deploy the container in production:

   a. Exit and delete the containers that currently run an older version of PAS for OpenEdge.

   b. Use the custom container image to create and run containers.

You can combine this workflow with the Rolling Update functionality in orchestration software such as Kubernetes to achieve minimal downtime.

8

Security Considerations

Security for PAS for OpenEdge in a Docker environment is related to the file system and the HTTPS functionality.

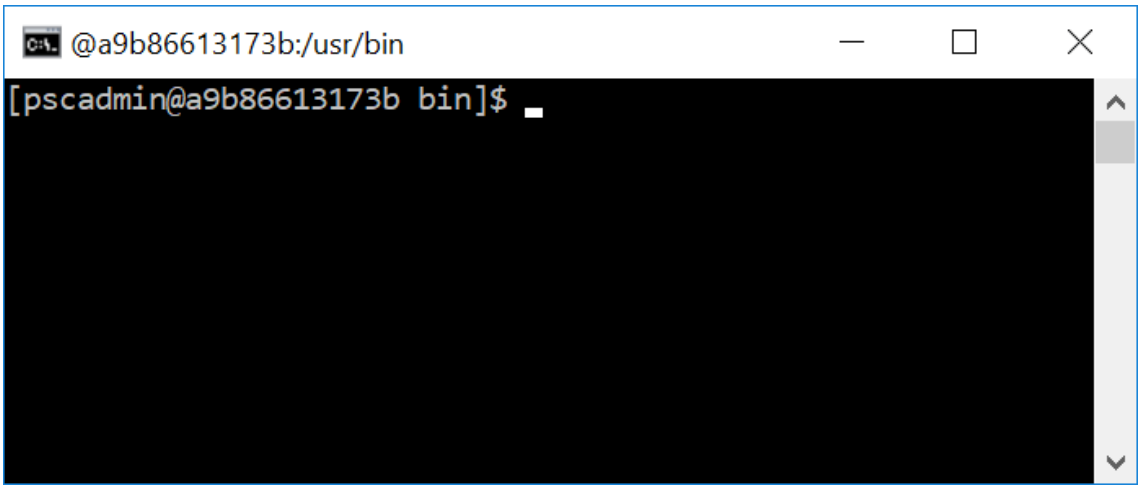
For details, see the following topics:

- File system security
- Directory and File Permissions
- Restrict client-server communication with HTTPs
- Install the SSL/TLS certificate

File system security

The following file system security features are implemented for the PAS for OpenEdge Docker container image:

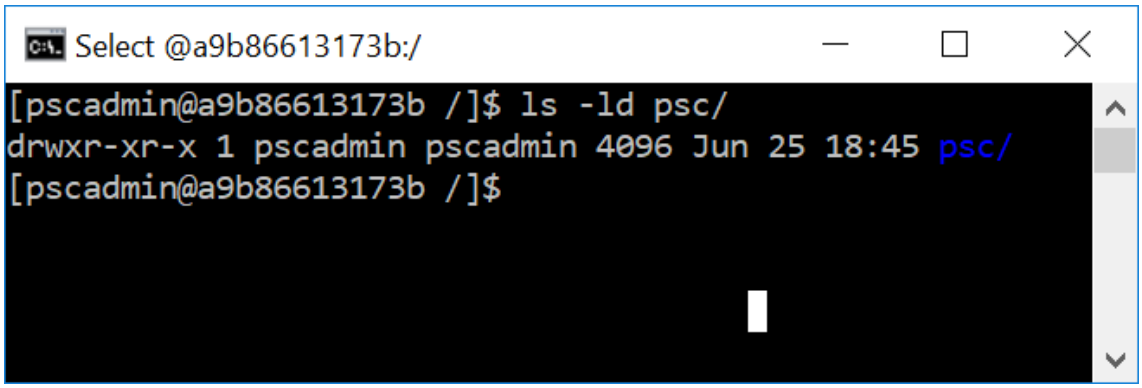
- The PAS for OpenEdge Docker container image creates the `pscadmin` user and group accounts.
- By default, you enter a container as a `pscadmin` user:



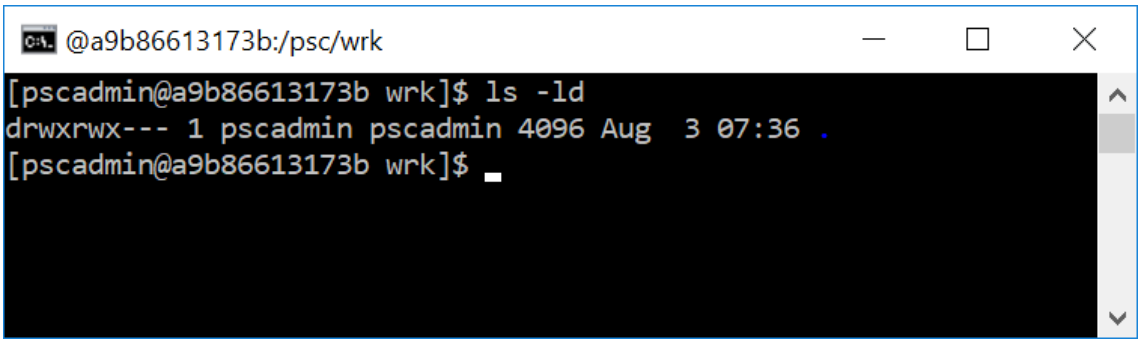
- The PAS for OpenEdge installation directory inside a Docker container is owned by `pscadmin`.

Directory and File Permissions

- For the `psc` directory:
  - The `pscadmin` user has `read (r)`, `write (w)`, and `execute (x)` permissions.
  - The `pscadmin` group has `read` and `execute` permissions.
  - All other users have `read` and `execute` permissions.

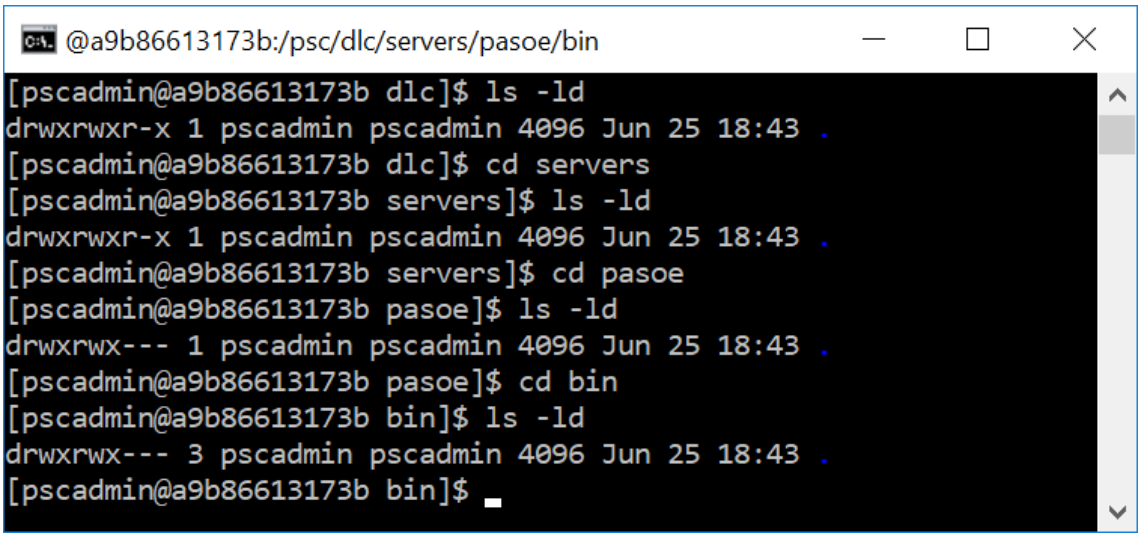


- Permissions are set differently for `psc/wrk`:



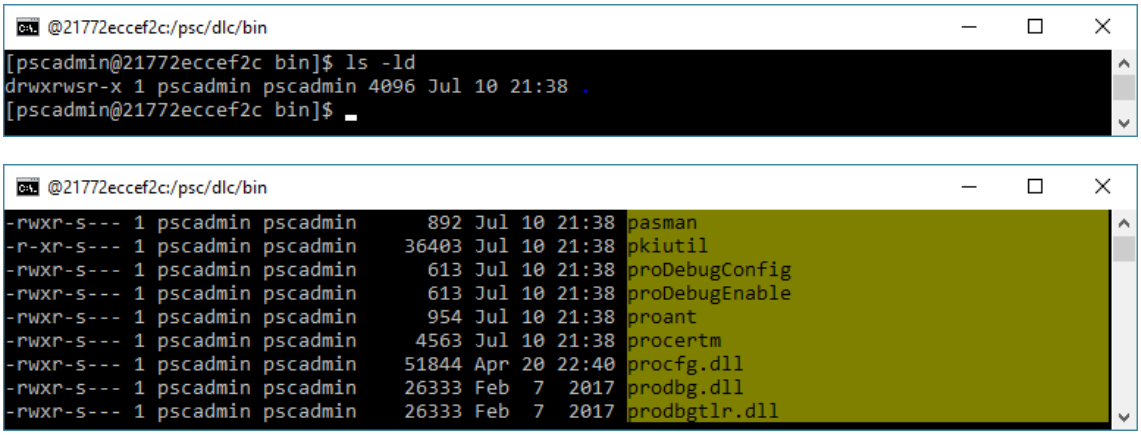
The `pscadmin` user and the `pscadmin` group both have `read`, `write`, and `execute` permissions. All others have none.

- The image below shows permissions for `dlc`, `dlc/servers`, `dlc/servers/pasoe`, and `dlc/servers/pasoe/bin`:



Note: Permissions for `$DLC/bin` may change as the Docker support for PAS for OpenEdge evolves.

- The special permission, `setgid`, is set on `DLC/bin`:



Notice the `s` instead of the executable `x` in the group permissions for the directory `DLC/bin` and for files in it. It indicates that `setgid` is set.

- The application files that are copied into a custom container image should be owned by `pscadmin`. Otherwise, they cannot be accessed in the container.

Restrict client-server communication with HTTPs

By default, the PAS for OpenEdge Docker container image has HTTPs enabled and HTTP disabled. The default value of the HTTPs port is 8811, this setting can be changed if desired. You can also turn on the HTTP functionality, though the use of HTTP is not recommended in secure environments.

To change instance specific settings, edit the files in the `conf` (`CATALINA_BASE/conf`) directory of the server instance.

For example: To turn on HTTP for an instance, edit the server.xml file in /instance/conf.

To change settings at the server level, edit the files in the `conf` (`CATALINA_HOME/conf`) directory of the server instance.

For example: to turn on HTTP for every instance that you create in future, edit the `server.xml` file in `/pasoe-install/pasoe/conf`.

For more information, see HTTPS Support.

Install the SSL/TLS certificate

Every new instance has a self-signed certificate, `PAS-default-cert`, in the `conf` folder. Use this certificate to test the HTTPs functionality of the server instance.

For production purposes it is required that you obtain a certificate from a third-party Certificate Authority (CA). To copy the certificate into your `conf` folder, add a `copy` command in your Dockerfile or use the `docker cp` command.

The steps to generate a certificate are the same for a PAS for OpenEdge container and a non-container instance.

Note: In the Dockerfile, add the command for copying the appropriate SSL/TLS certificate into a custom container image of PAS for OpenEdge. When you build a custom container image using the Dockerfile, the created image will have the certificate in the `conf` folder.

See also:

- Securing PAS for OpenEdge communications