Andrey Lucyan Nascimento Paiva

Portabilidade de Aplicação em Tecnologias O-PAS

Campina Grande, PB

2026

Andrey Lucyan Nascimento Paiva

Portabilidade de Aplicação em Tecnologias O-PAS

Trabalho de Conclusão de Curso (TCC) submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Orientador: George Acioli Júnior, D.Sc.

Campina Grande, PB

2026

Andrey Lucyan Nascimento Paiva

Portabilidade de Aplicação em Tecnologias O-PAS

Trabalho de Conclusão de Curso (TCC) submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado. Campina Grande, PB, _____/_____/_____.

George Acioli Júnior, D.Sc.
Orientador

Eisenhawer de Moura Fernandes, D.Sc.
Convidado

Campina Grande, PB
2026

Em memória de Lucineide Nascimento Souza Sobral, minha tia e segunda mãe que nos deixou no decorrer de minha graduação.

Agradecimentos

Agradeço, primeiramente, à minha família, que sempre me apoiou e fez questão que eu continuasse perseverando independende de que tipo de dificuldade se encontrava no caminho. Meus pais, Anchieta e Luciane, e meu irmão, Anchieta Filho. Sem eles nada seria possível.

Agradeço também a minhas tias, irmãs de minha mãe, Luciene e Lucineide, que também serviram como mães para mim, além de terem criado meus primos que levo para a vida como irmãos, Vinícius, Aleff, Kianne, Lucas e Lorena.

Agradeço também aos grandes amigos que fiz desde que me mudei para Campina Grande, em especial Bruno Nascimento, Caio Oliveira, Jan Filho, Leonardo Pessôa, Lucas Lobo, Micael Espínola, Victor Klein e Yuri Dantas, que nos momentos bons e ruins do curso, sempre estiveram por perto para compartilhar dores e risadas.

Por fim, agradeço a toda a equipe do LIEC, em especial: Davi Ramalho, Danilo de Oliveira, Francinildo Figueiredo, João Pedro Dantas, Mateus Figueiredo e Yan Abrantes, que me receberam de braços abertos durante meu último ano de graduação, mesmo eu sendo aluno de outra ênfase, e me proporcionaram diversos bons momentos e novos aprendizados.

Obrigado.

"Amai a vossos inimigos, bendizei os que vos maldizem, fazei bem aos que vos odeiam, e orai pelos que vos maltratam e vos perseguem.

...

Pois, se amardes os que vos amam, que galardão tereis?",

(Mateus 5:44, 46)

Resumo

Com a chegada da indústria 4.0, métodos que possibilitem a reutilização de lógicas de controle são cada vez mais procurados. Dentre esses métodos, a criação de aplicações de controle portáteis, ou seja, aplicações capazes de serem transferidas de ambiente de execução sem perdas significativas de desempenho. Nesse trabalho, foi utilizada a tecnologia Docker para implementar uma simulação feita utilizando os blocos de função definidos na IEC 61131-3. Os resultados obtidos mostram que o uso médio de CPU de um contêiner não supera 1 núcleo do processador, enquanto o atraso médio de comunicação é inferior a 1 ms, tornando a solução confiável e de baixo custo computacional.

Palavras-chave: O-PAS, Docker, Portabilidade de aplicação, análise de desempenho de contêiner, IEC 61131-3, CODESYS

Abstract

With the arrival of the industry 4.0, methods that enable the reuse control logics have become increasingly in demand. Among these methods, the creation of portable control applications, that is, applications capable of being transferred between different execution environments without significative losses. In this work, the Docker technology was used to implement a simulation using the function blocks defined in IEC 61131-3. The obtained results show that the average CPU usage of a container does not exceed one processor core, while the average communication delay is inferior to 1 ms, showing that the solution is reliable, and of low computational cost.

Key-words: O-PAS, Docker, Application portability, container performance analysis, IEC 61131-3, CODESYS

Lista de ilustrações

Figura 1 – Arquitetura de referência do O-PAS 6
Figura 2 – Ilustração da execução de um contêiner em uma máquina virtual . . . 8
Figura 3 – a) Antes da edição; b) Depois da edição 12
Figura 4 – Obtenção do IP do terminal Debian 13
Figura 5 – CODESYS conectado ao terminal Debian via SSH 14
Figura 6 – a) Configuração de vEdge; b) Configuração de vPLC1 14
Figura 7 – Configuração do Gateway 15
Figura 8 – Configuração final do dispositivo no CODESYS 15
Figura 9 – Visualização do conteúdo do contêiner no IP informado 16
Figura 10 – Uso de CPU de vPLC1 durante um intervalo de 5 minutos 17
Figura 11 – Atraso de comunicação entre vPLC1 e o servidor durante um intervalo de 5 minutos 17

Lista de tabelas

Tabela 1 – Estatísticas de Uso de CPU 18
Tabela 2 – Estatísticas de Latência de Rede 18

Lista de abreviaturas e siglas

O-PAS	*Open Process Automation Standard*
OPAF	*Open Process Automation Forum*
DCN	*Distributed Control Node*
IEC	*International Electrotechnical Commission*
WSL	*Windows Subsystem for Linux*
SSH	*Secure Shell*
DCN	*Distributed Control Node*
OCF	*O-PAS Connectivity Framework*
ACP	*Advanced Computing Platform*
OPC UA	*Open Platform Communications Unified Architecture*
OCI	*O-PAS Communication Interface*
CLP	*Controlador Lógico Programável*
LD	*Ladder Diagram*
ST	*Structured Text*
FBD	*Function Block Diagram*
SFC	*Sequential Function Chart*
IL	*Instruction List*
IP	*Internet Protocol*

Sumário

1 INTRODUÇÃO 1
1.1 Objetivo 1
1.2 Organização do trabalho 1

2 FUNDAMENTAÇÃO TEÓRICA 3
2.1 O-PAS 3
2.1.1 Arquitetura Geral do O-PAS 6
2.2 Docker e conteineres 7
2.3 Codesys e IEC 61131-3 7

3 PROVA DE CONCEITO 10
3.1 Configurando o ambiente WSL 10
3.2 Configurando e executando o CODESYS com Docker 13
3.3 Análise de desempenho do Contêiner 16

4 CONCLUSÃO 19
4.1 Perspectivas futuras 19

REFERÊNCIAS 20

1 Introdução

Historicamente, o controle de plantas industriais tem sido dominado por sistemas de automação proprietários e rigidamente integrados. Nesses ecossistemas, o hardware e o software são inseparáveis, criando o fenômeno do vendor lock-in, onde a substituição de um componente ou a atualização de uma funcionalidade exige investimentos elevados e dependência exclusiva de um único fornecedor.

Com o advento da indústria 4.0 e aumento cada vez maior da complexidade dos sistemas de automação industrial, cria-se uma crescente demanda por flexibilidade, interoperabilidade, portabilidade e eficiência nas plantas industriais. Este cenário cria um crescente interesse pela padronização das técnicas utilizadas para o desenvolvimento de tais sistemas. O *Open Process Automation Standard* (O-PAS), desenvolvido pelo *Open Process Automation Forum* (OPAF). O objetivo central do O-PAS é definir uma arquitetura de referência baseada em padrões abertos, que permita a construção de sistemas de controle escaláveis, interoperáveis e, crucialmente, portáteis. [1]

Para atingir tais requisitos, uma característica necessária para as aplicações em conformidade com o O-PAS é a portabilidade, que refere-se à capacidade de mover aplicações de controle, configurações e lógica de um *Distributed Control Node* (DCN) para outro, independentemente do fabricante do hardware ou do sistema operacional utilizados.

Neste trabalho é feita a validação experimental da portabilidade de uma aplicação em conformidade o padrão O-PAS, de forma a analisar o passo a passo para implementação e os efeitos da conteinerização em um sistema de controle desenvolvido segundo a norma de número 61131-3 da *International Electrotechnical Commission* (IEC).

1.1 Objetivo

O principal objetivo desse trabalho é validar conceitualmente e experimentalmente a implementação de uma aplicação de controle desenvolvida de acordo com a IEC 61131-3 em um contêiner, destacando as métricas de consumo de processamento e memória utilizados durante o processo.

1.2 Organização do trabalho

Este trabalho está dividido em 5 capítulos, incluindo esta seção introdutória.

O capítulo 2 apresenta a fundamentação teórica necessária para o entendimento das atividades desenvolvidas, abordando os conceitos centrais sobre o padrão O-PAS e seu

papel na indústria; Docker e o conceito de contêinerização [2]; a norma IEC 61131-3 e a ferramenta CODESYS [3].

O capítulo 3 ilustra o processo de execução do projeto, detalhando a configuração do ambiente de execução, utilizando o *Windows Subsystem for Linux* (WSL) [4], a implantação do contêiner e a comunicação com o CODESYS via protocolo *Secure Shell* (SSH), além de mostrar a análise de desempenho dos contêineres instânciados.

Por fim, o capítulo 4 sintetiza as conclusões que podem ser retiradas do trabalho, além de discutir possíveis direções para futura expansão das ideias apresentadas.

2 Fundamentação Teórica

Este capítulo detalha os conceitos e ferramentas necessárias para entendimento da arquitetura utilizada e experimentos realizados. São apresentadas as características principais do padrão O-PAS e seu papel na indústria, além de uma visão geral sobre o conceito e aplicação da tecnologia Docker, juntamente à principal norma de controle utilizada na indústria contemporânea, a IEC 61131-3, finalizando com a apresentação da aplicação utilzada como estudo de caso no trabalho.

2.1 O-PAS

O *Open Process Automation Standard* (O-PAS) é uma iniciativa da indústria que tem o objetivo de aumentar a vida útil dos sistemas de controle a partir da unificação dos sistemas em uma arquitetura padronizada, aberta, segura e interoperável [5].

A motivação principal para a criação do O-PAS é de reduzir ou até eliminar pontos de insatisfação que os usuários têm em relação aos sistemas de controle atuais, tais como:

- Falta de interoperabilidade entre equipamentos de diferentes fabricantes
- Inabilidade de reutilizar aplicações e configurações entre aparelhos de diferentes marcas ou até entre equipamentos de diferentes gerações da mesma fornecedora
- Componentes fechados e proprietários, o que, por vezes, gerava necessidade de substituir todo o sistema em caso fosse preciso trocar apenas um equipamento
- Falta de mecanismos nativos de Cibersegurança nos sistemas

O O-PAS é desenvolvido segundo o método de "padrão de padrões", onde uma arquitetura unificada é construída a partir de padrões já existentes na indústria, sendo um novo padrão desenvolvido apenas caso não exista nenhum padrão aceitável. Dessa forma, o O-PAS é dividido em 7 partes, cada uma referenciando seus respectivos padrões.

- **Parte 1:** Arquitetura técnica (IEC 62264)
- **Parte 2:** Segurança (IEC 62443)
- **Parte 3:** Perfis
- **Parte 4:** Conectividade (IEC 62541)
- **Parte 5:** Gerenciamento de sistema (Redfish)

- **Parte 6:** Modelos de Informação e Troca (IEC 62714, IEC 62682, IEC 61131 e IEC 61499)
- **Parte 7:** Plataforma física

A segunda edição do padrão define e descreve 23 atributos-chave necessários para a implementação de um sistema de automação aberto, distribuído e seguro: [6]

- **Capacidade de descobrimento:** Possibilidade de encontrar um item e entender seu modelo de troca de informações e capabilidades.
- **Compatibilidade:** Capacidade de uma unidade funcional de atender aos requerimentos de determinada interface sem necessidade de modificações.
- **Confiabilidade:** Capacidade de um sistema ou componente de executar suas funcionalidades necessárias sob determinadas condições por um intervalo de tempo desejado.
- **Configurabilidade:** Capacidade de ajuste dentro de um conjunto finito de parâmetros de forma a se adequar às necessidades do usuário.
- **Conformidade com os padrões:** Satisfação de 100% dos requisitos de conformidade de qualquer perfil do O-PAS citado ou implementado no sistema.
- **Disponibilidade:** Grau ao qual um sistema ou componente é operacional e acessível quando requisitado.
- **Escalabilidade:** Grau ao qual um sistema pode ter a capacidade ajustada para atender aos requerimentos do projeto.
- **Evolutividade:** Possibilidade de um sistema ou componente de sofrer melhorias ou ter novas capacidades adicionadas.
- **Flexibilidade:** Facilidade com a qual um sistema ou componente pode ser modificado para uso em aplicações ou ambientes diferentes dos quais foi projetado para atuar.
- **Gerenciabilidade:** Capacidade de monitorar e controlar o estado de integridade dos componentes.
- **Intercambialidade:** Capacidade de um componente ser substituido por outro, possivelmente de outra fabricante, sem adicionais modificações ao sistema.
- **Interoperabilidade:** Capacidade de dois ou mais sistemas ou componentes trocarem informações entre si e utilizarem efetivamente tais informações.

- **Manutenibilidade:** Facilidade com a qual hardware, software ou um componente podem ser reparados ou modificados para:
  - Alterar ou adicionar capabilidades;
  - Corrigir falhas ou defeitos;
  - Melhorar performance ou outros atributos;
  - Adaptação a um novo ambiente.
- **Modularidade:** Grau em que um sistema ou aplicação é composto por componentes discretos de tal forma que mudanças em um desses componentes tenham impacto mínimo nos outros.
- **Portabilidade:** Facilidade com a qual um sistema ou componente pode ser transferido de um ambiente de hardware ou software para outro.
- **Rastreabilidade:** O grau em que uma relação pode ser estabelecida entre dois ou mais produtos do processo de desenvolvimento, sendo ela uma associação identificável entre duas ou mais entidades lógicas, tais como requisitos, elementos do sistema, verificações ou tarefas.
- **Resiliência:** Capacidade de um sistema ou componente de realizar suas funções apesar de eventos adversos.
- **Reusabilidade:** O grau em que um componente pode ser utilizado em mais de um sistema ou na construção de diferentes sistemas.
- **Segurabilidade:** Capacidade de um componente ou sistema de ser protegido contra acessos ou modificações não autorizadas.
- **Segurança:** Expectativa de que um componente ou sistema, sob condições definidas, não crie uma situação em que vidas humanas, saúde, propriedade ou o ambiente estejam sob perigo.
- **Testabilidade:** O grau em que um teste objetivo e viável pode ser projetado para determinar se um requisito foi atendido.
- **Usabilidade:** A facilidade com a qual um usuário específico pode aprender a operar, gerar entradas para ou interpretar saídas de um sistema ou componente.
- **Viabilidade econômica:** Característica do projeto expressa como a solução que atinge as necessidades ou requisitos de um cliente dentro de um preço aceitável.

2.1.1 Arquitetura Geral do O-PAS

A arquitetura geral do O-PAS é ilustrada na Figura 1. Nela podemos ver claramente os três principais componentes da arquitetura O-PAS, os *Distributed Control Nodes* (DCNs), conectados via *O-PAS Connectivity Framework* (OCF) e a *Advanced Computing Platform* (ACF). O OCF é um framework de rede de dados que fornece um serviço lógico de troca de dados a todos os nós participantes, utilizando mecanismos de transporte OPC UA e o modelo de informações do O-PAS. A função do OCF é de garantir determinismo e qualidade de serviço da forma necessária para que ocorram trocas de dados visando a execução lógica para o controle distribuido [6].

Um DCN é qualquer dispositivo em conformidade com o O-PAS que execute funções de entrada/saída ou aplicações definidas pelo usuário, que se conecte fisicamente e em termos de dados ao OCF e que pode ou não ser conectado a outros dispositivos físicos [5]. Um DCN virtual é um DCN rodando numa plataforma física virtualizada.

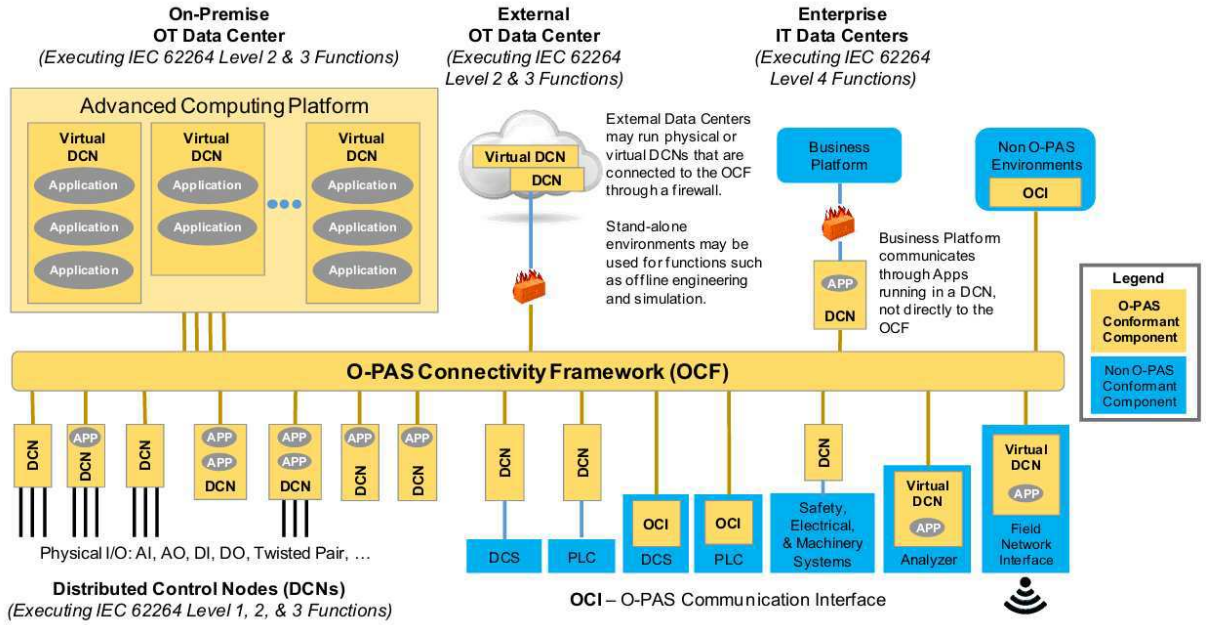


Figura 1 – Arquitetura de referência do O-PAS

Fonte: [6]

A ACP é uma plataforma de computação que fornece determinismo, baixa latência e confiabilidade necessários para executar aplicações ou serviços que requerem mais recursos que os tipicamente disponíveis em DCNs [6]. As ACPs podem ser utilizadas para executar aplicações que não são facilmente ou eficientemente distribuídas e fornecem abstrações como máquinas virtuais onde funções de DCN podem ser realizadas. Aplicações executadas em uma ACP de comunicam com outras aplicações O-PAS via *O-PAS Communication Interface* (OCI).

2.2 Docker e conteineres

Docker é uma plataforma aberta para desenvolvimento, envio e execução de aplicações [7], executadas em ambientes isolados chamados Contêineres, os quais são unidades padronizadas de software que envelopam o código de uma aplicação junto a todas suas dependências, como bibliotecas e arquivos de configuração. Para isso é utilizada uma funcionalidade chave do *kernel Linux*: sempre que um contêiner é executado, o Docker cria um conjunto de *namespaces* específicos para o funcionamento daquele contêiner. Dessa forma, um usuário pode executar a mesma aplicação em outro ambiente virtual de maneira confiável e instantânea, sem ser necessária a instalação de dependências adicionais.

O Docker é dividido em *Docker Engine* e *Docker hub*. O Engine contém as ferramentas necessárias para criar e conteinerizar as aplicações. Já o Docker hub é uma biblioteca para compartilhar e organizar tais aplicações (de forma similar ao git), chamadas *Imagens*. Todas as imagens publicadas no Docker hub podem ser baixadas para qualquer máquina e executadas sem pacotes adicionais. As informações necessárias para a criação das imagens estão armazenadas na *Dockerfile*, que é um arquivo de texto contendo os comandos necessários para a criação do ambiente da imagem.

Para a virtualização, o Docker utiliza uma rede isolada padronizada enquanto o contêiner é executado, a não ser que seja especificada uma rede pelo usuário. Durante a execução de um contêiner, é estabelecida uma conexão Ethernet virtual, de forma que contêineres na mesma rede possam se comunicar entre si, enquanto isolados do *host* e de outros contêineres não conectados nessa rede. A cada contêiner é atribuido um endereço de IP virtual e uma ou mais portas correspondentes ao seu acesso [2]. A figura 2 ilustra um exemplo da instânciação de um contêiner, a partir de uma dockerfile, gera-se a imagem correspondente, que é subida ao Docker hub. Posteriormente, essa imagem pode ser carregada em uma máquina virtual e executada num ambiente diferente da qual foi gerada.

2.3 Codesys e IEC 61131-3

A norma de número 61131 da *International Electrotechnical Comission* (IEC) estabelece as diretrizes para padronização dos Controladores Lógicos Programáveis (CLPs). Anteriormente a sua adoção na indústria, os ambientes de programação dos CLPs eram proprietários de cada fabricante, o que dificultava ou até impossibilitava a migração de uma aplicação de um sistema para outro. A norma é dividida em 10 partes [8]:

- **Parte 1:** Visão geral
- **Parte 2:** Hardware

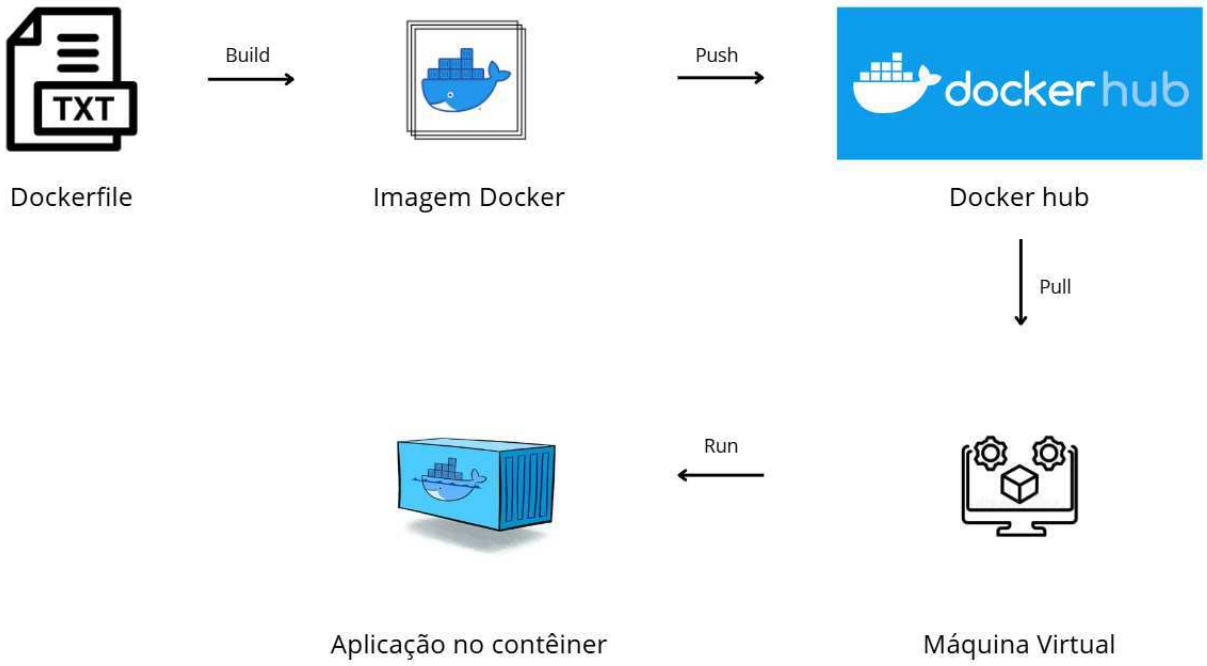


Figura 2 – Ilustração da execução de um contêiner em uma máquina virtual

Fonte: Adaptado de [7] e Google Images

- **Parte 3:** Linguagens de programação
- **Parte 4:** Guias de usuário
- **Parte 5:** Comunicação
- **Parte 6:** Segurança funcional
- **Parte 7:** Programação de controle *fuzzy*
- **Parte 8:** Guias para aplicação e implementação das linguagens de programação
- **Parte 9:** Interface de comunicação digital *single-drop* para pequenos sensores e atuadores
- **Parte 10:** Formato de troca PLCOpen XML

Dentre estas, a terceira parte, é referente às linguagens de programação utilizadas em CLPs na indústria. A norma padroniza 4 linguagens de programação para construção da lógica das aplicações:

- *Ladder Diagram* (LD)
- *Structured Text* (ST)

- *Function Block Diagram* (FBD)
- *Sequential Function Chart* (SFC)
- *Instruction List* (IL)

A linguagem utilizada para a implementação desse trabalho é o FBD, a qual utiliza blocos de função para a construção lógica da aplicação. Blocos de função são unidades lógicas que encapsulam um ou mais algoritmos e podem ser utilizados para resolver de pequenos a grandes problemas de automação de forma reutilizável e interoperável.

Como ambiente de desenvolvimento, foi utilizado o software CODESYS [3], que é uma das plataformas de software mais difundidas na automação industrial moderna para implementação da IEC 61131-3. A ferramenta fornece, entre outras funcionalidades, o desacoplamento entre hardware e software, a partir de uma arquitetura que permite a instalação do sistema de execução (runtime) em diversas plataformas, como o Linux e dispositivos embarcados.

3 Prova de Conceito

Este capítulo mostra a metodologia para execução de uma aplicação em conformidade com o O-PAS no ambiente Docker, de forma que o sucesso na implementação da configuração proposta funcione como prova de conceito para a portabilidade de aplicações em tecnologias O-PAS, uma vez que foi discutido o conceito de portabilidade segundo a norma e o Docker atinge os requisitos de abstração e isolamento necessários para tal característica.

3.1 Configurando o ambiente WSL

O *Windows Subsystem for Linux* (WSL) [4] é uma ferramenta que permite ao usuário rodar um ambiente Linux numa máquina Windows, sem a necessidade de uma máquina virtual ou *Dual boot*. Para utilização do CODESYS, a distro utilizada é o Debian [9]. No terminal do Debian, os comandos a serem executados inicialmente são:

```
sudo apt-get update && \
sudo apt-get install -y ca-certificates curl gnupg python3 && \
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o/etc/apt/
keyrings/docker.asc && \
sudo chmod a+r /etc/apt/keyrings/docker.asc && \
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
docker.asc] https://download.docker.com/linux/debian$(./etc/os-release
&& echo "$VERSION_CODENAME") stable" | sudo tee/etc/apt/sources.list.d
/docker.list > /dev/null
```

Essa sequência faz a preparação do Debian para reconhecer o Docker e receber seus pacotes.

No trecho:

```
sudo apt-get update && \
sudo apt-get install -y ca-certificates curl gnupg python3 && \
```

O comando **update** atualiza todos os pacotes do sistema para as versões mais atuais disponíveis.
O comando **install -y** instala as seguintes ferramentas:

- **ca-certificates:** Permite que o Debian consiga verificar conexões HTTPS
- **curl:** Permite que o Debian consiga baixar dados da internet
- **gnupg:** Permite que o Debian verifique chaves de criptografia
- **python3:** Pacote necessário para o funcionamento do ambiente

No trecho:

```
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o/etc/apt/
keyrings/docker.asc && \
sudo chmod a+r /etc/apt/keyrings/docker.asc && \
```

O comando **curl** faz a instalação da chave pública do Docker e a armazena em `/etc/apt /keyrings/docker.asc`, enquanto o comando **chmod a+r** permite que o gerenciador de pacotes tenha permissão para fazer a leitura dessa chave.

No trecho:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
docker.asc] https://download.docker.com/linux/debian$(./etc/os-release
&& echo "$VERSION_CODENAME") stable" | sudo tee/etc/apt/sources.list.d
/docker.list > /dev/null
```

A parte `arch=$(dpkg --print-architecture)` detecta automaticamente a arquitetura do processador e utiliza como argumento na função. A parte `signed-by=/etc/apt/keyrings/docker.asc]` dá o endereço da chave pública adquirida anteriormente para realizar a validação do repositório.

`https://download.docker.com/linux/debian$(./etc/os-release && echo "$VERSION_CODENAME") stable"` é o endereço do repositório do docker para Debian, a ser baixado para a versão específica do sistema. Por fim, `sudo tee/etc/apt/sources.list.d/docker.list > /dev/null` altera o arquivo de configuração sem exibir nada no terminal.

Após configurado o ambiente do Docker, são instalados os os arquivos necessários para executar o Docker Engine e habilitada a comunicação com esse ambiente via *Secure Shell* (SSH). Isso é feito a partir dos seguintes comandos:

```
sudo apt-get update && \
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin openssh-server
```

Esse comando indica que o motor a ser utilizado é o **docker-ce** (Docker Community Edition), com a interface de linhas de comando **docker-ce-cli**; o **containerid.io** é responsável por gerenciar os contêineres. O trecho **docker-buildx-plugin** instala o plugin Buildx e o trecho **docker-compose-plugin** instala o plugin Docker Compose. Por fim a parte **openssh-server** permite que esse servidor seja acessado remotamente por qualquer outra máquina na rede.

Em seguida, é executado o comando

```
sudo nano /etc/ssh/sshd_config
```

Nesse arquivo, é localizada a linha contendo `#PermitRootLogin prohibit-password` (Figura 3-a), ela deve ser descomentada e editada para `PermitRootLogin yes` (Figura 3-b). Esse comando faz com que todos os usuários que saibam a senha do usuário root consigam se conectar à rede.

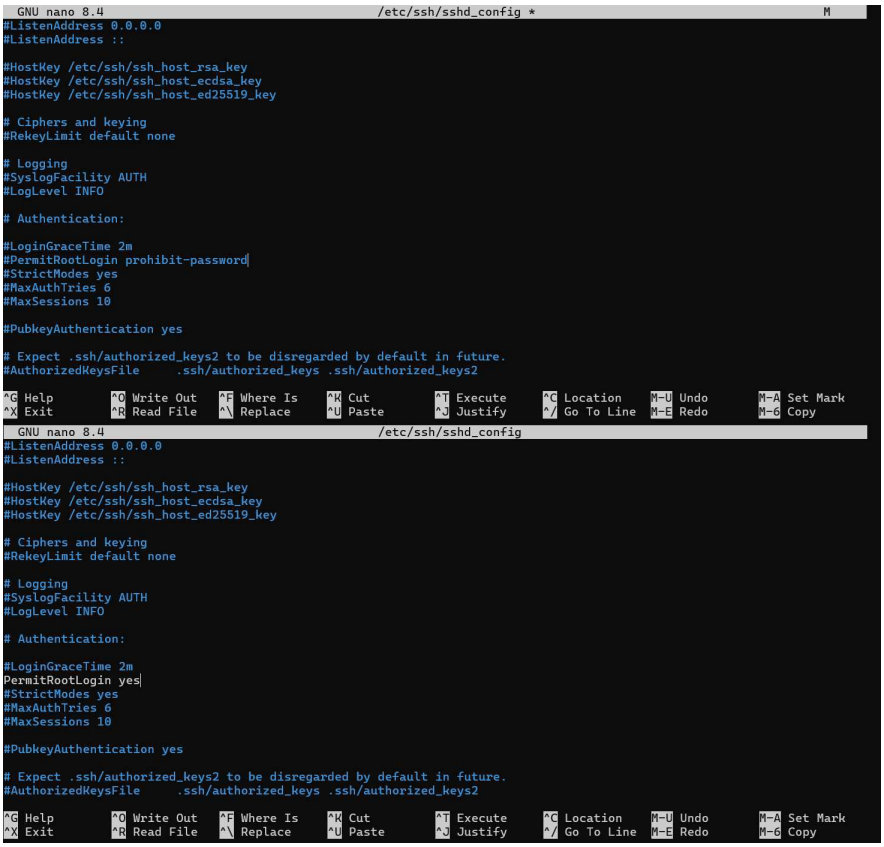


Figura 3 – a) Antes da edição; b) Depois da edição

Fonte: Autoria própria

Por fim, é executado o comando

```
sudo systemctl enable --now ssh
```

Que habilita a comunicação via SSH com o Debian e a inicializa sempre que o computador for ligado.

3.2 Configurando e executando o CODESYS com Docker

O CODESYS nativamente possui pacotes adicionais fornecidos pela própria distribuidora que tornam o software compatível com Docker. No *CODESYS installer*, os seguintes pacotes são necessários:

- CODESYS LicenseServer for Linux SL
- CODESYS Virtual Control for Linux SL
- CODESYS Control SL Deploy Tool

Para encontrar o IP gerado para o Debian no WSL, é executado o seguinte comando no Windows (Figura 4):

```
wsl -d Debian hostname -I
```

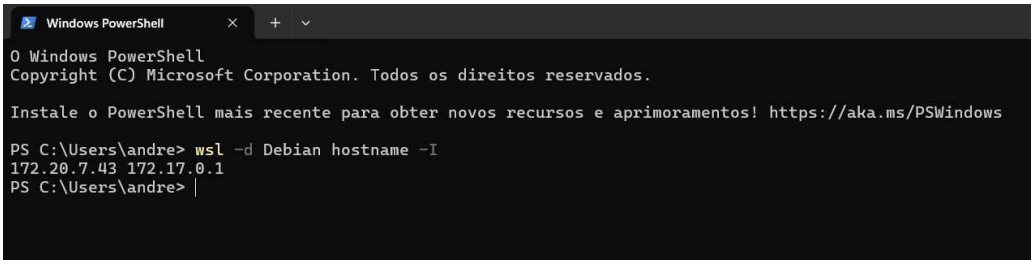


Figura 4 – Obtenção do IP do terminal Debian

Fonte: Autoria Própria

Adquirido o IP 172.20.7.43, ele deve ser colocado no CODESYS em Ferramentas -> Deploy Control SL, juntamente com o usuário e a senha de root criados na instânciação do WSL (Figura 5).

Na aba de Implementação, no menu esquerdo, são adicionados o pacote **CODESYS Server for Linux SL** e as imagens **CODESYS Virtual Control for Linux SL** e **CODESYS Virtual Edge Gateway for Linux**. O gateway será chamado **vEdge**, enquanto o runtime será chamado **vPLC1**. Na aba de Operação, a configuração de vEdge



Figura 5 – CODESYS conectado ao terminal Debian via SSH

Fonte: Autoria própria

é mostrada na Figura 6-a enquanto a configuração de vPLC1 é mostrada na Figura 6-b. A porta escolhida para acesso ao contêiner foi a 8080 e para acesso ao gateway 1217.

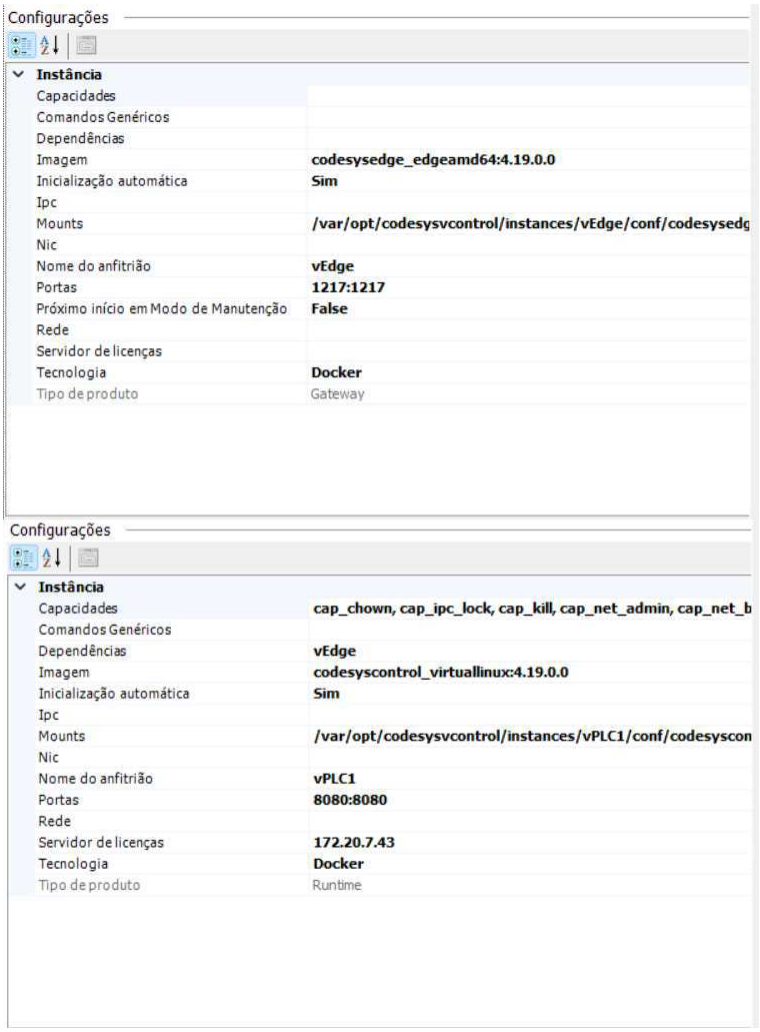


Figura 6 – a) Configuração de vEdge; b) Configuração de vPLC1

Fonte: Autoria própria

No projeto do CODESYS, abrindo as configurações do dispositivo e selecionando Gateway, pode ser feita a conexão a um novo gateway, nesse caso, o vEdge (Figura 7).

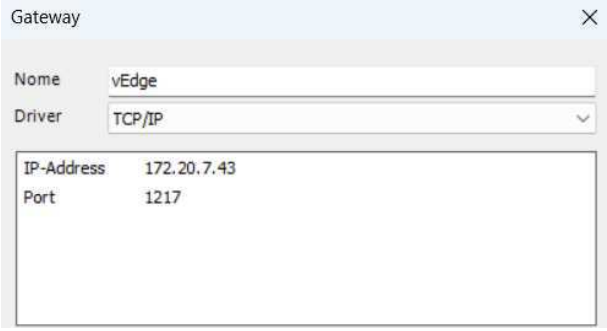


Figura 7 – Configuração do Gateway

Fonte: Autoria própria

No botão ao lado, em Escanear rede, selecionamos o vPLC1. Se tudo estiver correto, irão aparecer dois círculos verdes ao lado do gateway e do PLC na rede do CODESYS (Figura 8).

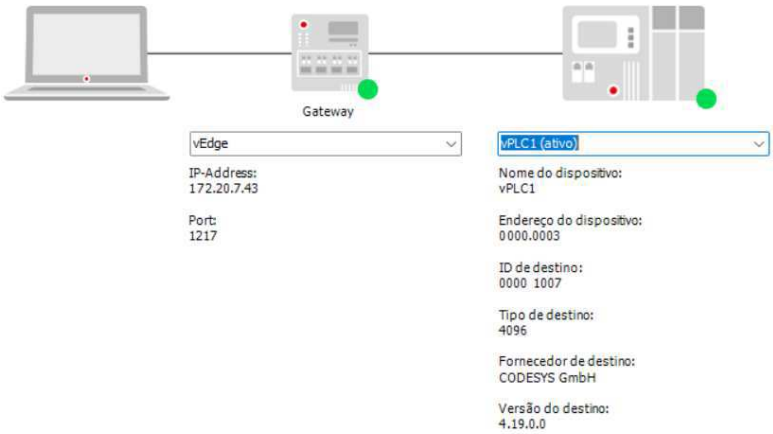


Figura 8 – Configuração final do dispositivo no CODESYS

Fonte: Autoria própria

Ao acessar o endereço do gateway, especificando a porta de interesse para visualização do CLP virtual, **172.20.7.43:8080**, é possível acessar o conteúdo do contêiner a partir de qualquer dispositivo conectado na rede (Figura 9). Neste trabalho, a aplicação escolhida para portabilidade por meio da contêinerização foi um projeto de estudo disponibilizado pela OPAF no Gitlab [10].

Dessa forma, ao implementar uma aplicação em conformidade com o padrão O-PAS e aplicar os conceitos estudados de Docker, é possível atingir os requisitos de abstração e isolamento necessários para a portabilidade de aplicações em tecnologias O-PAS

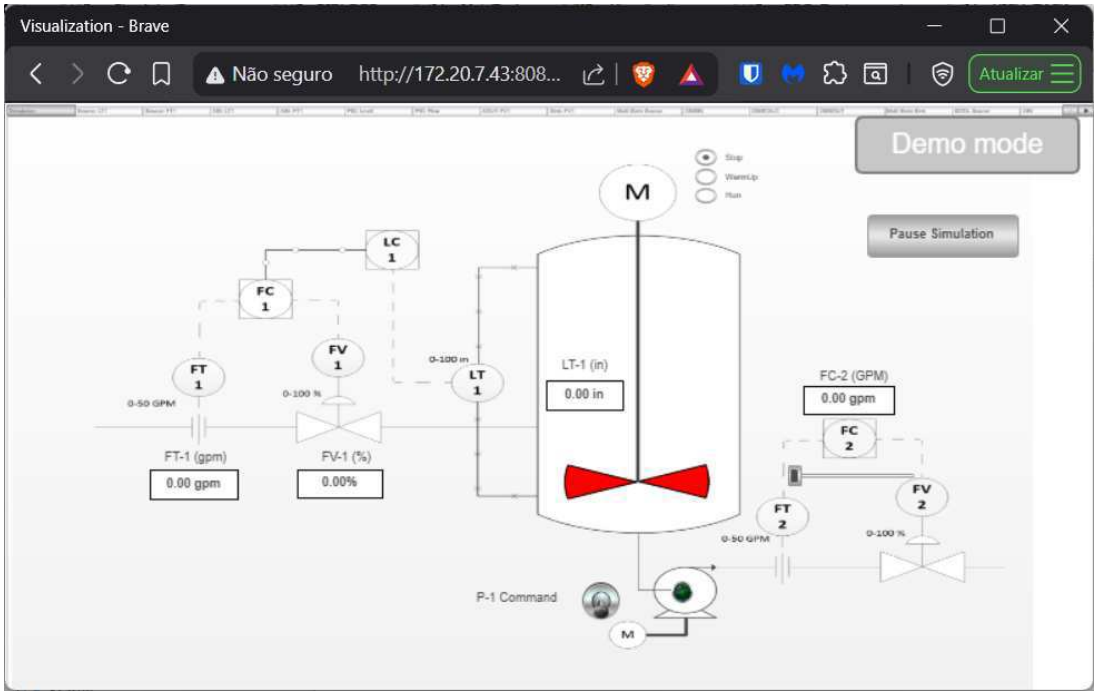


Figura 9 – Visualização do conteúdo do contêiner no IP informado

Fonte: Autoria própria

3.3 Análise de desempenho do Contêiner

Tão importante quanto analisar a possibilidade de implementar uma aplicação portátil, é analisar a viabilidade de tal prática. Para isso, serão analisados os parâmetros de uso de CPU e memória da máquina, em conjunto com o atraso na comunicação entre o vPLC e o servidor. Para observar o uso de CPU e memória, utilizamos o comando:

```
sudo timeout 300s docker stats >> usocpu.txt
```

Para adquirir o atraso de comunicação entre vPLC1 e o servidor, o comando utilizado é:

```
sudo timeout 300s docker exec -it vPLC1 ping 172.17.0.3 >> latencias.txt
```

Após um breve tratamento de dados, a fim de obter os valores no formato desejado para a análise, é possível gerar gráficos do comportamento do uso da CPU (Figura 10) e da latência(Figura) por vPLC1 durante um intervalo de 5 minutos.

A partir da leitura dos dados, é possível obter, também, outras métricas de qualidade sobre o uso de CPU (Tabela 1) e a latência (Tabela 2)

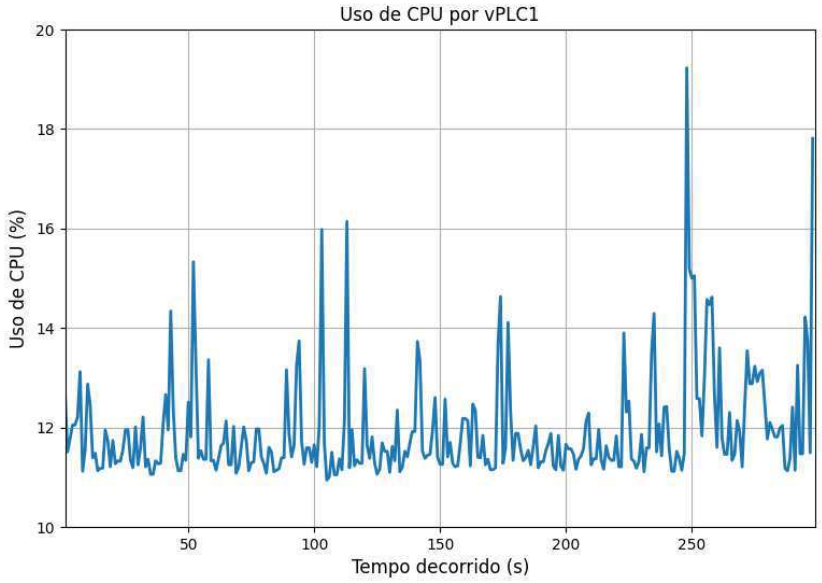


Figura 10 – Uso de CPU de vPLC1 durante um intervalo de 5 minutos

Fonte: Autoria própria

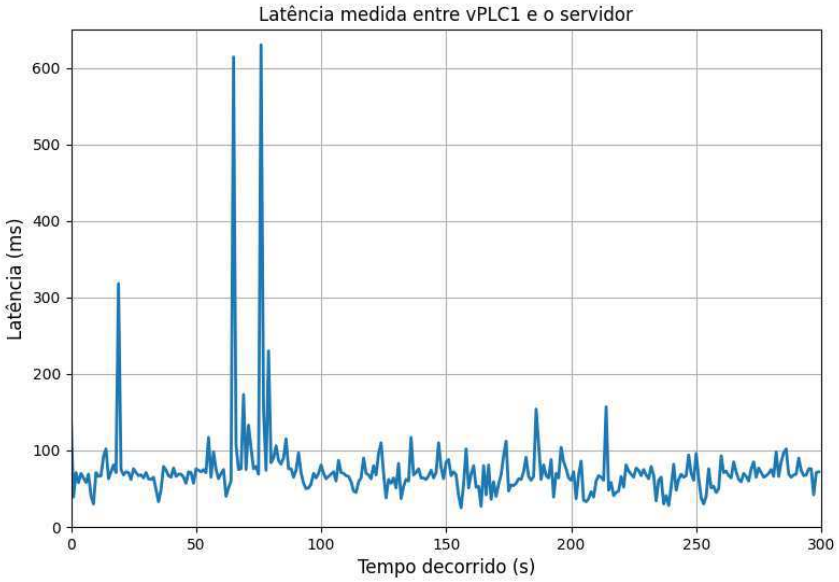


Figura 11 – Atraso de comunicação entre vPLC1 e o servidor durante um intervalo de 5 minutos

Fonte: Autoria própria

Métrica de Uso de CPU	**Valor (%)**
Uso de CPU Mínimo	0,08
Uso de CPU Médio	11,87
Uso de CPU Máximo	19,23
Desvio Padrão	1,26

Tabela 1 – Estatísticas de Uso de CPU

Fonte: Autoria própria

Métrica de Latência	**Valor (ms)**
Latência Mínima	0,025
Latência Média	0,074
Latência Máxima	0,630
Desvio Padrão	0,052

Tabela 2 – Estatísticas de Latência de Rede

Fonte: Autoria própria

4 Conclusão

A partir dos resultados obtidos na Prova de Conceito juntamente com a análise de desempenho, é possível confirmar a viabilidade do uso da tecnologia de contêinerização para atingir os requisitos de portabilidade do padrão O-PAS, uma vez que o Docker consegue atingir os níveis de abstração necessários para garantir a portabilidade sem que haja uma necessidade muito elevada de poder computacional, nem uma perda de comunicação significativa. O uso médio de CPU de 11,87% durante a execução mostra que, em média, o contêiner ocupa menos de 1 núcleo de um processador, enquanto a latência média medida de 0,074 ms confirma a viabilidade do uso da tecnologia Docker para aplicações em tempo real, visto que um atraso na ordem de 70 microssegundos indica que o sistema, com melhores refinamentos, pode ser utilizado até em aplicações críticas.

4.1 Perspectivas futuras

O estudo desenvolvido nesse trabalho abre perspectivas para adicionais explorações sobre o tema, que incluem, mas não se limitam a:

- Estudo de viabilidade de Docker com outras normas de controle, como por exemplo a IEC 61499.
- Estudo de desempenho de uma aplicação contêinerizada rodando em um Raspberry Pi.
- Estudo da implementação com mais dispositivos, de forma a analisar o impacto causado nas métricas de desempenho causado por um maior número de dispositivos.
- Estudo de métodos de cibersegurança voltados para aplicações em Docker, uma vez que a aplicação é visível a todos os dispositivos conectados na rede.

Referências

[1] P. Jhunjhunwala, S. S. Bitar, K. Zhukovskii, U. D. Atmojo, and V. Vyatkin, "Interoperability in software-defined process automation using the open process automation standard and iec 61499: Adapter connections of iec 61499 and opas enable plug-and-play integration," *IEEE Industrial Electronics Magazine*, vol. 18, no. 4, pp. 4–15, 2024. Citado na página 1.

[2] M. Sollfrank, F. Loch, S. Denteneer, and B. Vogel-Heuser, "Evaluating docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3566–3576, 2021. Citado 2 vezes nas páginas 2 e 7.

[3] CODESYS GmbH, *CODESYS Development System V3 User Manual*. CODESYS GmbH, Kempten, Germany, 2024. Acessado em: 25 nov. 2025. Citado 2 vezes nas páginas 2 e 9.

[4] Microsoft, "What is the windows subsystem for linux?." https://learn.microsoft.com/en-us/windows/wsl/about, 2024. Acesso em: 08 mar. 2026. Citado 2 vezes nas páginas 2 e 10.

[5] R. D. Bartusiak, S. Bitar, D. L. DeBari, B. G. Houk, D. Stevens, B. Fitzpatrick, and P. Sloan, "Open process automation: A standards-based, open, secure, interoperable process control architecture," *Control Engineering Practice*, vol. 121, p. 105034, 2022. Citado 2 vezes nas páginas 3 e 6.

[6] The Open Group, "O-pas™ standard, 2nd edition: Part 1 - technical architecture overview (informative), version 2.1," Standard C230-1, The Open Group, Reading, UK, 2025. Document Number: 1-957866-10-9. Citado 2 vezes nas páginas 4 e 6.

[7] Docker Documentation, "Docker overview." `https://docs.docker.com/get-started/docker-overview/`, 2024. Acesso em: 10 mar. 2026. Citado 2 vezes nas páginas 7 e 8.

[8] PLCopen, "Iec 61131-3: a standard programming resource," tech. rep., PLCopen, 2016. Citado na página 7.

[9] Projeto Debian, "Debian – o sistema operacional universal," 2026. Acesso em: 16 mar. 2026. Citado na página 10.

[10] The Open Group, "O-pas function block standard set: Part 6.4 information models," 2026. Acesso em: 16 mar. 2026. Citado na página 15.