IMPLEMENTAÇÃO DE UM AMBIENTE KUBERNETES PARA MONITORAMENTO DE SENSORES: UMA SOLUÇÃO ESCALÁVEL E EFICIENTE

JOÃO EMANUEL RICCI BARBOSA

ENGENHARIA DE SOFTWARE
SENAI Londrina

FACULDADE DE TECNOLOGIA SENAI LONDRINA
Rua Belém, 844 | 86026-000 | Londrina PR (43) 3294-5100

Sumário

1. Introdução

"[...] para que haja automação industrial é, antes de tudo, preciso que haja indústria, e ainda processos automáticos auto controláveis." (SILVEIRA, 2003 p.1).

A crescente demanda por automação e monitoramento em tempo real exige soluções robustas e escaláveis para a gestão de dados.

Atualmente na indústria, são utilizados de CLPs (Controladores Lógicos Programáveis), que atuam como controladores nos processos industriais, funcionando como um sistema de controle e monitoramento, recebendo dados de sensores e outros dispositivos de entrada, e posteriormente processando esses dados e controlando dispositivos de saída, uma abordagem robusta, duradoura e confiável, porém, a escalabilidade, flexibilidade e integração dos dados, são limitados.

Este artigo descreve a implementação de um ambiente Kubernetes, utilizando de técnicas como *Fog Computing*, *Cloud Computing* e mensageria, para coletar, gerenciar e analisar dados de sensores de máquinas de forma eficiente e acessível, argumentando sobre benefícios e desafios da implementação.

1.1 Problemática

A implementação de CLPs, é atualmente, a primeira escolha de engenheiros e especialistas em automação industrial, primeiramente pela sua alta confiança, durabilidade e robustez no recebimento, processamento e transmissão de dados, porém, são limitados nos quesitos de escalabilidade, flexibilidade e integração de dados, o que pode fazer com que, conforme a indústria cresça, mais e mais alterações deverão ser feitas o que gera aumento de custo. Por exemplo, dado um cenário em que uma empresa, no momento, de médio porte, que possui 15 máquinas e uma variação de 3 a 5 sensores por máquina, uma das linhas mais populares do mercado é a Siemens SIMATIC S7-1200. Contando com 14 *inputs* e 10 *outputs* digitais, e com capacidade de encaixar mais módulos, seu valor é de aproximadamente R$1.500,00. Se, em um futuro próximo, a empresa obtiver mais máquinas e aumentar sua variação de sensores, adquirir mais módulos ou, o controlador deverá ser substituído, e o custo aumentado.

1.2 Objetivos

O objetivo deste artigo, é discutir acerca do uso de containers em ambientes IoT e nuvem na manufatura, seus benefícios e desafios relacionados ao uso de tecnologias de *Containers*, como Docker e Kubernetes, para implementar soluções nesses ambientes. Juntamente com

a escrita do artigo, foi proposto pelo professor orientador, o desenvolvimento de um sistema que implemente a solução da problemática apresentada.

2. Referencial teórico

2.1 Arquiteturas tradicionais (CLPs e IHMs)

Arquiteturas baseadas em CLPs são amplamente utilizadas em ambientes de manufatura. Segundo Eueliton M et al. (2016), o CLP (Controlador Lógico Programável) tem o papel de receber essas instruções e fazer com que o sistema funcione conforme descrito nas linhas de programação. Todo esse processo, de receber sinais de sensores e atuar em saídas, foi possível visualizar em uma tela na IHM. Toda informação monitorada na IHM foi enviada a um servidor, compartilhando com OPC cliente, tendo como principal objetivo descrever informações diárias no software Excel, disponibilizando-as em planilhas e gráficos. Para que tudo isso ocorresse, foi preciso aprofundar-se nos processos de configuração do CLP e IHM.

Segundo Oliveira (2024. p. 18): “o controlador lógico programável (CLP) emerge como o cérebro da automação, assumindo a responsabilidade de controlar e coordenar todas as operações na célula. Recebendo informações dos sensores, processando dados e emitindo comandos aos atuadores, inversores e controladores, o CLP é fundamental para assegurar a sincronia e eficiência do processo. Sua capacidade de tomar decisões em tempo real contribui para a agilidade e adaptação do sistema às demandas variáveis do ambiente”.

De acordo com Siemens (2024): “Os controladores SIMATIC S7-1200 são a escolha ideal quando se trata de executar tarefas de automação de forma flexível e eficiente na faixa de desempenho inferior a médio. Eles apresentam uma ampla gama de funções tecnológicas e comunicação integrada, bem como design especialmente compacto e que economiza espaço.”

Atualmente no mercado, majoritariamente, os CLPs possuem alguma forma de IHC, permitindo realizar cadastros de processos, verificar status das sequências automáticas, observar os estados dos equipamentos, executar comandos manuais nos equipamentos e dispositivos quando necessário, verificar entradas e saídas digitais, certificar valores de processo, cancelar sequências em execução, buscar históricos de entrada, acompanhar falhas e intertravamentos ativos (individualizado em telas separadas) e monitorar o processo como um todo, tudo através de interação humana com alguma interface interativa, como por exemplo, uma tela touchscreen (OLIVEIRA, JOÃO PEDRO B, 2024. p. 19).

De acordo com João Pedro B. (2024. p. 26), a fase de desenvolvimento da automação com o CLP, envolveu a elaboração da lógica de programação, contemplando, as operações que o mesmo irá controlar, incluindo também, a arquitetura da rede industrial para a conexão entre o CLP, a IHM, sensores, dentre outros dispositivos e máquinas.

Dentre as limitações e desafios da implementação e utilização das arquiteturas baseadas em CLPs, temos:

1. **Escalabilidade restrita:** Os CLPs são projetados para gerenciar um número limitado de entradas e saídas físicas. Ampliar a capacidade requer a adição de módulos ou novos CLPs, o que pode aumentar significativamente os custos e a complexidade. Em sistemas que precisam crescer rapidamente, essa restrição torna-se uma barreira à expansão escalável.
2. **Dependência de hardware proprietário:** Muitos fabricantes de CLPs utilizam hardwares e softwares proprietários, o que limita a interoperabilidade e encarece atualizações e manutenções. Isso reduz a flexibilidade para integrar novas tecnologias ou soluções de outros fornecedores.
3. **Capacidade Computacional Limitada:** CLPs possuem recursos computacionais restritos e são otimizados para tarefas específicas, como controle lógico sequencial e tempo real. A execução de algoritmos complexos de análise, aprendizado de máquina ou big data é inviável diretamente em CLPs.
4. **Conectividade limitada:** Muitos CLPs antigos ou convencionais não possuem suporte nativo para protocolos modernos de comunicação em rede, como MQTT, *Websockets* ou integrações baseadas em *APIs REST*. Isso dificulta a integração com sistemas em nuvem e arquiteturas IoT.
5. **Falta de flexibilidade para atualizações:** A atualização de firmware ou programas em CLPs requer frequentemente a parada do sistema, o que pode gerar tempo de inatividade. Afeta a continuidade operacional em indústrias onde o tempo de funcionamento é crítico.
6. **Dificuldade em lidar com dados em tempo real e *Big Data*:** Embora sejam bons para controle em tempo real, CLPs não são projetados para gerenciar grandes volumes de dados ou realizar análises avançadas. Isso limita sua capacidade de integrar-se a soluções modernas de análise preditiva e manutenção baseada em dados.
7. **Custos de manutenção e ciclo de vida:** A substituição de componentes, como módulos ou controladores completos, pode ser cara e está diretamente vinculada ao fornecedor. A vida útil reduzida de certos componentes eletrônicos pode aumentar os custos operacionais a longo prazo.

8. **Baixa Adaptabilidade a Mudanças:** Adaptações em arquiteturas baseadas em CLPs, como a mudança de lógica ou adição de novas funcionalidades, frequentemente demandam alterações físicas ou reconfigurações complexas. Afeta negativamente a agilidade na implementação de novos requisitos.
9. **Complexidade na integração com tecnologias modernas:** CLPs não são projetados para trabalhar nativamente com arquiteturas distribuídas, como ambientes de containers, ou integrar-se diretamente com soluções em nuvem. A adaptação para ambientes modernos exige *Gateways* ou soluções intermediárias, o que aumenta a complexidade do sistema.

Em suma, esses desafios evidenciam que, embora os CLPs sejam adequados para aplicações industriais tradicionais, sua arquitetura não é ideal para sistemas escaláveis, distribuídos e baseados em tecnologias modernas. A transição para soluções como *Containers* e Kubernetes permite superar muitas dessas limitações, oferecendo maior flexibilidade, escalabilidade e capacidade de integração com tecnologias emergentes.

2.2 Arquitetura de containers

1. Containers

Segundo MORAIS L. da SILVA (2019, p. 12), os *Containers* são uma tecnologia que permite empacotar e isolar aplicações dentro de um sistema operacional, ou seja, possibilita rodar múltiplos espaços de usuários isolados em um mesmo *Kernel*.

Seu foco é isolar processos e recursos em vez de emular todo um servidor físico. Do ponto de vista de uma aplicação, cada container é um sistema operacional completo.

Containers rodam diretamente no sistema operacional compartilhando seus recursos de forma otimizada. Por serem mais otimizados em relação a servidores virtuais, são leves e iniciam rápido.

Containers podem ser vistos como ferramentas mais flexíveis para empacotamento, entrega e orquestração de serviços e aplicativos de software. Por serem uma tecnologia

recente, permitem melhor portabilidade e interoperabilidade, o que vem contribuindo para uma mudança substancial nos métodos de desenvolvimento e publicação de aplicativos.

Os containers são baseados em uma arquitetura de virtualização de nível de sistema operacional, onde múltiplos containers podem compartilhar o mesmo *Kernel* do sistema operacional, mas ainda assim operar de maneira isolada. Essa abordagem é mais leve e eficiente em comparação com máquinas virtuais, já que não exige a duplicação de um sistema operacional completo. É popular em ambientes de desenvolvimento e operações (DevOps), pois facilita a implantação, escalabilidade e gestão de aplicativos distribuídos.

Em resumo, containers ajudam a garantir que uma aplicação funcione de maneira idêntica em diferentes sistemas, simplificando o processo de desenvolvimento e operações, oferecendo várias vantagens, como portabilidade, permitindo que aplicações rodem de forma consistente em diferentes ambientes. Eficientes em recursos, pois compartilham o mesmo *Kernel* do sistema operacional, tornando-os mais leves que máquinas virtuais. Oferecem isolamento, o que aumenta a segurança e reduz o impacto de falhas. Também facilitam a escalabilidade, permitindo a rápida expansão de recursos conforme a demanda, especialmente em arquiteturas de microsserviços. Além disso, contribuem para ciclos de desenvolvimento mais rápidos, com implantação e testes ágeis, e são ideais para práticas de integração e entrega contínua (CI/CD).

2. Docker

Acerca de ferramentas famosas para desenvolver e entregar software, temos várias ferramentas, porém, a que será utilizada na solução proposta é Docker. Por definição, Docker é uma plataforma que permite empacotar e executar aplicativos de forma isolada, em ambientes chamados containers. Cada container contém tudo o que é necessário para rodar uma aplicação — como código, bibliotecas e dependências — garantindo que ela funcione da mesma maneira em diferentes sistemas. As vantagens do Docker incluem a portabilidade, facilidade de implantação e consistência entre os ambientes de desenvolvimento, teste e produção.

3. Kubernetes

A respeito de gerenciamento e orquestração de sistemas distribuídos temos Kubernetes, que por outro lado, é um sistema de orquestração para *Containers*. Ele gerencia a execução, escalabilidade e distribuição de containers em um *Cluster* de servidores, automatizando tarefas como balanceamento de carga, recuperação de falhas e atualizações contínuas. As vantagens do Kubernetes incluem a automação e o gerenciamento eficiente de grandes volumes de containers, proporcionando escalabilidade, alta disponibilidade e uma infraestrutura mais resiliente.

4. Vantagens da arquitetura de containers

Arquiteturas de containers e Kubernetes oferecem uma série de vantagens sobre as arquiteturas tradicionais de automação industrial, como CLPs (Controladores Lógicos Programáveis) e IHMs (Interfaces Homem-Máquina). Em vez de depender de hardware específico e de sistemas fechados, como ocorre nas soluções tradicionais, o uso de containers permite que aplicativos e serviços sejam isolados, portáveis e facilmente escaláveis, o que é crucial em ambientes industriais dinâmicos. Containers podem ser distribuídos em diferentes máquinas e atualizados sem afetar o funcionamento de outros componentes, o que traz flexibilidade e agilidade para as operações industriais. Já o Kubernetes, como orquestrador, gerencia automaticamente a execução desses containers, garantindo alta disponibilidade, escalabilidade e recuperação rápida em caso de falhas, algo que seria complexo de se implementar com CLPs e IHMs tradicionais. Juntos, Docker e Kubernetes oferecem uma solução robusta para o desenvolvimento e a operação de aplicativos modernos em ambientes de nuvem ou servidores locais.

Enquanto as soluções convencionais exigem hardware específico e muitas vezes dependem de interações complexas e lentas para atualização de sistemas, a arquitetura de containers com Kubernetes permite que novos serviços sejam facilmente desenvolvidos, testados e implantados em minutos. Isso reduz custos operacionais, melhora a resiliência da infraestrutura e facilita a integração com outras tecnologias, como inteligência artificial e análise de dados em tempo real, algo que seria mais difícil de realizar em sistemas tradicionais de controle e

supervisão. Dessa forma, a combinação de containers e Kubernetes oferece uma maior flexibilidade, eficiência e inovação em comparação com as arquiteturas convencionais no setor industrial.

2.3 Tecnologias e infraestrutura

1. ***Fog* e *Cloud Computing***

Fog Computing e *Cloud Computing* são duas arquiteturas complementares que têm ganhado relevância no contexto de Internet das Coisas (IoT), com a crescente necessidade de processamento e armazenamento de grandes volumes de dados gerados por dispositivos conectados. A Cloud Computing, tradicionalmente associada a grandes centros de dados centralizados, oferece infraestrutura escalável e flexível, permitindo o processamento e armazenamento de dados em larga escala. Já a *Fog Computing* expande o conceito de computação em nuvem para a borda da rede, aproximando o processamento e análise de dados dos dispositivos finais, como sensores e microcontroladores.

Em contextos de IoT, a *Cloud Computing* permite o armazenamento massivo de dados e a execução de tarefas complexas de processamento em grandes servidores remotos. Isso é particularmente vantajoso em cenários onde a capacidade computacional necessária ultrapassa o que é viável em dispositivos com recursos limitados, como microcontroladores. Além disso, com a *Cloud Computing*, a gestão de dados em larga escala, incluindo *Big Data*, se torna mais eficiente, uma vez que a infraestrutura de nuvem oferece ferramentas avançadas de análise e aprendizado de máquina para tratar grandes volumes de informações de forma ágil e escalável.

Por outro lado, a *Fog Computing* contribui significativamente em contextos onde a latência precisa ser minimizada. Isso ocorre porque os dados são processados mais perto de sua origem, no que são chamados de "nós de borda", ao invés de depender de servidores remotos na nuvem. Em ambientes IoT, onde dispositivos geram dados em tempo real e a resposta imediata é muitas vezes crítica, a *Fog Computing* permite uma redução considerável na latência, além de permitir o processamento local de dados para reduzir a dependência de conexões com a nuvem e evitar sobrecarga nas redes.

2. Uso de *Fog* e *Cloud Computing* na arquitetura de *Containers*

Docker e Kubernetes desempenham um papel essencial tanto em ambientes de *Cloud* quanto de *Fog Computing*, pois proporcionam ferramentas para orquestrar, escalar e gerir containers, o que facilita o desenvolvimento e a implantação de aplicações distribuídas. Com o uso de containers, é possível isolar aplicações e serviços, garantindo maior flexibilidade e eficiência no gerenciamento de recursos computacionais, algo crucial em arquiteturas distribuídas que envolvem dispositivos IoT com diferentes capacidades.

A integração de tecnologias como Docker e Kubernetes com IoT, *Fog* e *Cloud Computing* também contribui para a criação de sistemas resilientes e escaláveis. Microcontroladores, que são componentes fundamentais em sistemas de IoT, podem ser facilmente integrados nesses ambientes, processando dados localmente (na borda) ou enviando dados para a nuvem para análises mais profundas. A utilização de *Big Data*, por sua vez, permite a análise em tempo real e a geração de insights a partir de grandes volumes de dados coletados, o que é essencial para aplicações como cidades inteligentes, saúde conectada e automação industrial.

Em suma, a combinação de *Fog* e *Cloud Computing* oferece um equilíbrio entre processamento de dados em tempo real e a capacidade de análise de grandes volumes de dados, enquanto tecnologias como Docker, Kubernetes e *Big Data* ampliam as possibilidades de escalabilidade, flexibilidade e eficiência na gestão de sistemas IoT complexos. Essas soluções são fundamentais para otimizar o desempenho e a tomada de decisões em ambientes altamente dinâmicos e distribuídos.

3. Big Data em sistemas distribuídos

Refere-se à abordagem de processar e analisar grandes volumes de dados de maneira eficiente e escalável, utilizando uma infraestrutura composta por múltiplos servidores ou nós que trabalham em conjunto. Em um sistema distribuído, os dados são armazenados e processados de forma paralela em diferentes máquinas ou clusters, permitindo lidar com a diversidade, volume e velocidade dos dados em tempo real. Esse tipo de arquitetura é especialmente útil em cenários onde a quantidade de dados gerados é muito grande para ser tratada

por um único servidor, ou quando há a necessidade de processamento em tempo real ou quase em tempo real.

A principal vantagem de um sistema distribuído para Big Data é a escalabilidade. Ao distribuir o processamento e o armazenamento de dados em vários nós, é possível adicionar recursos conforme a necessidade, sem comprometer o desempenho. Isso é especialmente importante em contextos como Internet das Coisas (IoT), redes sociais, e-commerce e outros setores que geram grandes quantidades de dados continuamente. A escalabilidade horizontal, em que novos servidores são adicionados ao sistema, é uma característica chave desta arquitetura, pois permite expandir a capacidade do sistema sem a necessidade de realizar grandes mudanças na infraestrutura existente.

Além da escalabilidade, os sistemas distribuídos para Big Data oferecem maior resiliência e disponibilidade. Em uma arquitetura distribuída, se um nó falhar, os outros nós podem assumir a carga de trabalho, o que torna o sistema mais robusto e menos suscetível a interrupções. Tecnologias como o Hadoop e o Apache Spark são exemplos de frameworks projetados para facilitar o processamento de grandes volumes de dados em ambientes distribuídos. O Hadoop, por exemplo, usa o HDFS (Hadoop Distributed File System) para armazenar dados de forma distribuída e o MapReduce para realizar o processamento paralelo. Já o Apache Spark, que é mais eficiente em operações de memória, permite o processamento de dados de forma rápida e com menor latência, sendo uma escolha popular em ambientes que exigem processamento em tempo real.

Outra característica importante de Big Data em sistemas distribuídos é a capacidade de lidar com dados em diferentes formatos e fontes. Dados estruturados, semiestruturados e não estruturados (como logs, imagens, vídeos, dados de sensores, etc.) podem ser armazenados e processados juntos em um sistema distribuído, utilizando ferramentas que permitem a integração e análise de dados heterogêneos. Isso é crucial em ambientes de IoT, onde dispositivos geram uma grande diversidade de dados em tempo real, e as informações precisam ser combinadas e analisadas de forma eficiente para gerar insights valiosos.

Em termos de desafios, um dos principais é a consistência e a coordenação entre os nós distribuídos. Em um

sistema distribuído, garantir que todos os nós tenham uma visão consistente dos dados é uma tarefa complexa, especialmente quando há atualizações simultâneas em várias partes do sistema. Protocolos como o *CAP Theorem* (Consistência, Disponibilidade e Tolerância à Partição) são fundamentais para entender as limitações e trade-offs envolvidos na construção de sistemas distribuídos. Em muitos casos, a escolha entre consistência e disponibilidade depende das necessidades específicas de cada aplicação.

Em resumo, Big Data em sistemas distribuídos permite a análise de grandes volumes de dados de forma escalável, resiliente e eficiente. Tecnologias como Hadoop, Spark, e a capacidade de lidar com dados em tempo real são fundamentais para esse tipo de arquitetura, que é amplamente utilizado em setores como IoT, saúde, finanças e e-commerce. No entanto, desafios relacionados à consistência, segurança e coordenação entre os nós precisam ser cuidadosamente gerenciados para garantir o bom funcionamento desses sistemas.

4. Microcontroladores e sua aplicação na arquitetura de containers

Microcontroladores são pequenos dispositivos eletrônicos com capacidade limitada de processamento e memória, geralmente usados para controlar sistemas embarcados e realizar tarefas específicas em uma variedade de aplicações, como automação, IoT, robótica, dispositivos vestíveis e outros sistemas de controle. Apesar de suas limitações de hardware, os microcontroladores desempenham um papel crucial na coleta de dados e na interação com o ambiente físico em sistemas distribuídos. A integração desses dispositivos com *Containers*, uma tecnologia tradicionalmente associada a sistemas mais robustos, tem se mostrado uma abordagem inovadora que combina os benefícios da flexibilidade e portabilidade dos containers com a eficiência e baixo consumo de energia dos microcontroladores.

Para que essa integração seja possível, as soluções de containers precisam ser adaptadas para funcionar em plataformas de baixo consumo de energia e com recursos limitados de hardware. Uma das abordagens é utilizar *micro containers*, que são versões reduzidas de containers tradicionais, projetadas para consumir menos recursos e oferecer uma sobrecarga mínima no sistema. Tecnologias como

K3s (uma versão reduzida do Kubernetes), para gerenciar containers em dispositivos IoT ou em redes de microcontroladores.

A utilização de containers em microcontroladores oferece diversas vantagens. Primeiramente, ela permite a portabilidade das aplicações: o mesmo container pode ser executado em dispositivos de diferentes fabricantes ou arquiteturas, simplificando o processo de desenvolvimento e *deployment*. Isso é particularmente importante em ambientes de IoT, onde pode haver uma grande diversidade de dispositivos e plataformas. Além disso, a utilização de containers permite a modularização das aplicações, de modo que diferentes componentes de software possam ser empacotados de forma independente, facilitando a atualização, manutenção e escalabilidade de sistemas distribuídos.

A integração de containers também melhora a gestão e orquestração de dispositivos. Em uma rede de microcontroladores distribuídos, a automação do *deployment* e a escalabilidade das aplicações tornam-se muito mais eficientes quando o software é encapsulado em containers. Ferramentas como o Docker Compose e Kubernetes permitem que aplicações sejam gerenciadas de forma declarativa, com a possibilidade de orquestrar, monitorar e escalar dispositivos e seus respectivos containers automaticamente. Essa abordagem torna o gerenciamento de redes de dispositivos mais ágil e menos suscetível a erros, além de facilitar a implementação de atualizações e patches de segurança em larga escala.

Contudo, a utilização de containers em microcontroladores também apresenta desafios. A sobrecarga de processamento e o consumo de recursos que os containers exigem podem ser um impeditivo, especialmente considerando que muitos microcontroladores têm CPU e memória limitadas. A adaptação das tecnologias de containers para funcionar de maneira eficiente em dispositivos com recursos tão restritos requer um design otimizado e leve, com sistemas operacionais e ferramentas que sejam compatíveis com a arquitetura de baixo nível dos microcontroladores. Além disso, a comunicação e o gerenciamento de redes de dispositivos com containers podem adicionar complexidade, principalmente em sistemas de grande escala ou em redes de dispositivos com conectividade intermitente ou limitada.

Em ambientes de IoT, onde microcontroladores estão frequentemente envolvidos, o uso de containers pode também facilitar a integração com sistemas baseados em nuvem ou com arquiteturas de *Fog Computing*, permitindo que os microcontroladores atuem como nós de borda que processam dados localmente em containers e, ao mesmo tempo, podem enviar ou sincronizar informações com servidores mais robustos ou sistemas de nuvem. Isso pode reduzir a latência na comunicação, melhorar a eficiência do uso de banda e otimizar o processamento em tempo real.

Em resumo, o uso de microcontroladores com *Containers*, é uma tendência emergente que promete trazer mais flexibilidade, portabilidade e eficiência para o desenvolvimento e gerenciamento de sistemas distribuídos, especialmente em aplicações de IoT. Embora existam desafios técnicos relacionados ao uso de containers em dispositivos com recursos limitados, a adaptação das tecnologias de containers para esses ambientes oferece benefícios significativos, como modularidade, orquestração e automação, permitindo que redes de dispositivos inteligentes e sistemas embarcados sejam mais fáceis de implementar, escalar e manter.

2.4 Princípios de Engenharia de Software

A adoção de boas práticas e padrões arquitetônicos é essencial no desenvolvimento de sistemas modernos, garantindo escalabilidade, manutenção e flexibilidade. Entre as abordagens mais relevantes destacam-se a Clean Architecture, o Repository Pattern, os princípios SOLID e o Domain-Driven Design (DDD).

1. **Clean Architecture**

   Proposta por Robert C. Martin, organiza o código em camadas de forma a separar responsabilidades e garantir que as regras de negócio, núcleo da aplicação, permaneçam independentes de *frameworks*, interfaces de usuário e bancos de dados. Essa separação promove um desacoplamento significativo, permitindo que mudanças tecnológicas em componentes periféricos não impactem as regras fundamentais do sistema, além de facilitar o processo de teste e manutenção.

2. **Repository Pattern**

Atua como uma camada de abstração no acesso a dados, isolando a lógica de negócios dos detalhes de implementação. Com ele, a aplicação interage com interfaces que representam repositórios, tornando possível modificar a fonte de dados — como trocar um banco relacional por um NoSQL — sem necessidade de alterações na lógica principal. Isso reduz o acoplamento, melhora a organização do código e facilita a adaptação a novos requisitos.

3. **SOLID**

Os princípios SOLID, também introduzidos por Robert C. Martin, oferecem diretrizes para o design de sistemas orientados a objetos. Cada princípio contribui para a criação de software mais modular e sustentável. O *Single Responsibility Principle* assegura que cada classe tenha uma única responsabilidade bem definida, enquanto o *Open*/*Closed Principle* recomenda que as classes estejam abertas para extensão, mas fechadas para modificação, garantindo maior segurança contra alterações indesejadas. O *Liskov Substitution Principle*, por sua vez, orienta que as subclasses possam substituir suas superclasses sem alterar o comportamento esperado. O *Interface Segregation Principle* sugere que classes não sejam forçadas a implementar interfaces que não utilizam, e o *Dependency Inversion Principle* reforça a ideia de que módulos de alto nível não devem depender de módulos de baixo nível, mas ambos devem depender de abstrações.

4. **Domain-Driven Design**

O *Domain-Driven Design* (DDD) propõe que o desenvolvimento do software seja orientado pelo domínio do problema, ou seja, pela lógica e pelos processos do negócio. Essa abordagem incentiva a colaboração contínua entre desenvolvedores e especialistas do domínio, garantindo que o software reflita com precisão os requisitos reais. O DDD também introduz conceitos como entidades, agregados e repositórios, que ajudam a estruturar o sistema de forma coerente. Entre suas principais vantagens estão a clareza na modelagem do domínio, a maior capacidade de lidar com sistemas complexos e a criação de um vocabulário comum entre as partes envolvidas.

Esses conceitos e tecnologias, quando aplicados em conjunto, oferecem um poderoso conjunto de ferramentas para o desenvolvimento de

sistemas modernos, equilibrando flexibilidade, robustez e alinhamento com as necessidades do negócio, principalmente para os profissionais de tecnologia, pois, ao participar de um projeto cujo código é baseado em um padrão de design, sua flexibilidade, manutenibilidade e a legibilidade do código, passam a ser drasticamente aumentada.

2.5 Escalabilidade e segurança em sistemas distribuídos e na arquitetura de *Containers*

A escalabilidade é um dos principais desafios e objetivos em soluções que combinam IoT, containers e sistemas distribuídos. Com o aumento exponencial no volume de dispositivos conectados e na quantidade de dados gerados, é essencial projetar arquiteturas capazes de crescer de maneira eficiente, tanto horizontal quanto verticalmente. A adoção de boas práticas em Kubernetes e em sistemas distribuídos, juntamente com medidas robustas de segurança, desempenha um papel central para garantir a robustez e a confiabilidade dessas soluções.

1. ***Containers*** **e Docker**

   A escalabilidade em ambientes de containers para IoT permite lidar com a carga variável de dispositivos e dados. Containers, como os criados pelo Docker, são leves e rápidos de iniciar, o que possibilita criar instâncias adicionais de serviços de forma quase instantânea em resposta ao aumento da demanda. A orquestração com Kubernetes torna essa escalabilidade gerenciável ao automatizar o provisionamento, balanceamento de carga e redistribuição de recursos. Em um cenário de IoT, onde dispositivos frequentemente se conectam e desconectam da rede, essa elasticidade é fundamental para assegurar que o sistema mantenha o desempenho.

2. **Escalabilidade horizontal e vertical, e segurança em sistemas distribuídos**

   A escalabilidade pode ser implementada de duas formas: horizontal e vertical. Na escalabilidade horizontal, novos nós ou instâncias de serviços são adicionados para atender ao aumento de demanda. Isso é especialmente relevante em sistemas IoT, pois cada dispositivo adicional que se conecta à rede representa um novo fluxo de dados que precisa ser processado. Kubernetes facilita esse tipo de escalabilidade com

ferramentas como auto scalers, que monitoram métricas de utilização (como CPU e memória) e criam ou removem instâncias automaticamente conforme necessário. Por outro lado, a escalabilidade vertical aumenta os recursos de uma única instância, como adicionando mais memória ou poder de processamento. Embora limitada pela capacidade física dos servidores, a escalabilidade vertical pode ser útil para cargas temporárias ou para otimizar o desempenho de serviços críticos.

Sistemas distribuídos e escaláveis também precisam considerar medidas de segurança robustas. Em ambientes Kubernetes, a segurança começa com o isolamento adequado dos containers por meio de namespaces, limitando o acesso entre serviços. O uso de políticas de segurança de *pod* (*PodSecurityPolicies*) garante que containers sejam executados com permissões mínimas, reduzindo a superfície de ataque. Além disso, é fundamental implementar práticas como a rotação frequente de segredos e tokens, uso de certificados TLS para comunicação entre serviços e a segmentação da rede com políticas específicas (*Network Policies*). Em sistemas IoT, onde dispositivos remotos podem ser alvos de ataques, o uso de autenticação mútua (*mutual TLS*) e a criptografia ponta a ponta para dados em trânsito são boas práticas essenciais.

A segurança e a privacidade dos dados também são questões importantes em sistemas distribuídos para Big Data. A distribuição dos dados em diferentes nós e locais pode tornar mais difícil garantir a integridade e a proteção contra acessos não autorizados. Tecnologias de criptografia e autenticação distribuída, além de políticas rigorosas de governança de dados, são essenciais para garantir que os dados sejam armazenados e processados de forma segura.

Outro aspecto crítico é a observabilidade e monitoramento de ambientes distribuídos. Ferramentas como Prometheus e Grafana permitem monitorar métricas em tempo real, ajudando a identificar gargalos ou anomalias que possam comprometer a escalabilidade ou a segurança. A integração com soluções como AWS GuardDuty ou Kubernetes Audit Logs oferece uma camada adicional de visibilidade sobre possíveis tentativas de ataque ou violações de segurança.

Boas práticas de segurança também se estendem ao ciclo de vida do software. O uso de imagens de container verificadas e atualizadas reduz a exposição a vulnerabilidades

conhecidas. Em sistemas IoT, onde os dispositivos podem operar em locais remotos, a capacidade de atualizar firmware e software de forma segura é essencial para prevenir ataques e corrigir falhas rapidamente.

A combinação de escalabilidade bem planejada e medidas de segurança robustas cria uma base confiável para sistemas IoT em ambientes de containers. Kubernetes, com sua capacidade de automação e resiliência, atua como o alicerce para gerenciar cargas variáveis e crescer de maneira eficiente, enquanto práticas de segurança bem estabelecidas garantem a proteção dos dados e dispositivos em todas as etapas da operação. Isso resulta em soluções capazes de atender às demandas crescentes do IoT sem comprometer desempenho ou segurança.

4. Caso de uso

O cenário proposto refere-se à implementação de uma solução de monitoramento e análise de dados de sensores em um ambiente de produção na indústria de manufatura. A empresa fictícia "ManuTech" busca otimizar suas operações e melhorar a manutenção dos equipamentos por meio do monitoramento em tempo real das máquinas de produção. A proposta envolve a criação de um ambiente baseado em Kubernetes para gerenciar e processar os dados gerados por sensores instalados nas máquinas.

A principal necessidade da "ManuTech" é ter uma solução escalável que permita a coleta, processamento e análise dos dados gerados pelos sensores, proporcionando insights detalhados sobre a performance das máquinas. A implementação deve garantir que a solução seja eficiente, capaz de detectar padrões e anomalias, além de permitir a geração de relatórios e alertas em tempo real para facilitar a tomada de decisões na manutenção e operação.

Acerca dos requisitos funcionais e não funcionais temos:

Requisitos funcionais

1.1. **Escalabilidade**: O sistema precisa ser capaz de escalar automaticamente para lidar com o aumento do número de sensores e volume de dados coletados.

1.2. **Interface de Usuário**: O sistema deve ter uma interface de dashboard que seja intuitiva, permitindo ao operador visualizar e filtrar os dados de forma eficiente.

1.3. **Gerenciamento de Mensagens**: O sistema de mensageria deve garantir a entrega eficiente e ordenada das mensagens enviadas pelos sensores, permitindo a comunicação contínua e sem falhas.

1.4. **Armazenamento e Processamento de Dados**: O servidor de coleta deve ser capaz de processar e armazenar grandes volumes de dados, proporcionando uma análise detalhada para a equipe de manutenção e analistas.

2. Requisitos não funcionais

2.1. **Escalabilidade Horizontal**: O ambiente deve ser projetado para escalar horizontalmente, aumentando a capacidade de processamento conforme o volume de dados cresce.

2.2. **Resiliência**: O sistema deve ser resiliente a falhas, com mecanismos de recuperação automática que garantam a continuidade do monitoramento e análise, mesmo em casos de falhas nos componentes do sistema.

2.3. **Redundância em mensageria**: O Apache Kafka é ideal para mensageria redundante devido à sua arquitetura distribuída, com particionamento e replicação de dados entre brokers para garantir alta disponibilidade e tolerância a falhas. Sua durabilidade e capacidade de recuperação automática asseguram confiabilidade e integridade mesmo em cenários críticos.

2.4. **Modularidade e Manutenibilidade**: O sistema deve seguir os princípios do DDD (Domain-Driven Design) e da Clean Architecture, garantindo baixo acoplamento e alta coesão entre os componentes. As entidades e serviços devem ser estruturados de forma clara, permitindo fácil atualização e extensão.

2.5. **Princípios de Design de Software**: Aplicar os princípios SOLID para promover um código robusto, reutilizável e com alta capacidade de evolução. O uso do Repository Pattern deve abstrair o acesso aos dados, promovendo desacoplamento entre a lógica de domínio e a camada de persistência.

2.6. **Desempenho e Baixa Latência**: Garantir que os dados sejam processados e entregues em tempo real, tanto no lado do cliente quanto no servidor, priorizando o uso de WebSockets para comunicação assíncrona e eficiente.

2.7. **Escalabilidade Horizontal**: Adotar a arquitetura de containers, onde cada microcontrolador possui seu próprio pod e conjunto de containers, garantindo isolamento e escalabilidade dos serviços para suportar alta carga e crescimento incremental.

2.8. **Confiabilidade e Redundância**: Implementar uma infraestrutura baseada em *Fog Computing* com Raspberry Pi para processar dados localmente e evitar dependência exclusiva de uma nuvem centralizada. O módulo WT32-ETH01 deve ser utilizado para comunicação via Ethernet, eliminando interferências comuns ao Wi-Fi.

2.9. **Observabilidade e Telemetria**: O sistema deve incluir logs detalhados, métricas e monitoramento distribuído para rastrear e diagnosticar falhas em tempo real.

2.10. **Segurança de Dados**: As informações sensíveis devem ser criptografadas utilizando AES-256 e SHA-256, tanto em trânsito quanto em repouso, com autenticação de dispositivos e controle de acesso baseado em usuários e permissões.

2.11. **Gestão Centralizada de Recursos**: O CRM deve fornecer uma interface intuitiva para cadastro, gerenciamento e visualização de entidades como clientes, empresas, sensores e microcontroladores. A visualização de dados em tempo real e o controle de entidades devem ser responsivos e de fácil uso.

2.12. **Portabilidade e Integração**: O sistema deve ser portável entre diferentes ambientes de Kubernetes, facilitando o gerenciamento e a orquestração dos containers e pods.

2.13. **Eficiência Energética**: Os dispositivos, incluindo microcontroladores e Raspberry Pi, devem operar de forma otimizada, reduzindo o consumo de energia, especialmente em ambientes sensíveis.

2.14. **Segurança**: A segurança do sistema é uma prioridade, e será implementada por meio da utilização de criptografia avançada AES-256 combinada com SHA-256, garantindo confidencialidade e integridade dos dados tanto em trânsito quanto em repouso. Além disso, mecanismos robustos de autenticação e controle de acesso baseados em permissões assegurarão que apenas usuários e dispositivos autorizados possam interagir com o sistema.

3. Contextualização

Na indústria de manufatura, o monitoramento constante da performance das máquinas é crucial para a otimização dos processos de produção e para a prevenção de falhas inesperadas nos equipamentos. Para a empresa "ManuTech", a coleta de dados em tempo real, com análise e visualização detalhada, possibilita identificar rapidamente problemas como vibrações ou temperaturas anormais, além de proporcionar a geração de relatórios e alertas para os responsáveis pela manutenção.

O uso de Kubernetes como plataforma para gerenciamento da infraestrutura permite que o sistema seja escalável e flexível, permitindo a adição de novos sensores e containers conforme a necessidade. O Apache Kafka, por sua vez, facilita a comunicação eficiente entre os sensores e o servidor de coleta de dados, garantindo que os dados sejam processados e armazenados sem perdas.

4. Descrição do ambiente implementado

O ambiente implementado para o caso de uso consiste em uma infraestrutura baseada em Kubernetes, que gerencia containers com sensores fictícios. Esses sensores, em containers, geram dados sobre a performance das máquinas de produção, como temperatura, vibração e outros indicadores de saúde das máquinas. Esses dados são enviados para o Kafka, que organiza e distribui as mensagens para o servidor de coleta, onde são processados e armazenados em uma base de dados para análise posterior.

O dashboard de visualização em tempo real permite ao operador de produção monitorar as condições atuais das máquinas e visualizar gráficos e métricas de desempenho. O operador pode filtrar os dados por sensor ou grupo de sensores e acompanhar a evolução dos parâmetros ao longo do tempo.

Além disso, a equipe de manutenção é alertada automaticamente quando os dados indicam falhas ou condições fora do padrão, como variações inesperadas nas medições dos sensores. A equipe de TI é responsável pela manutenção do ambiente Kubernetes e pela solução de problemas que possam surgir, como falhas de comunicação com o Kafka, garantindo a continuidade da coleta de dados.

O sistema é projetado para ser altamente escalável e resiliente, com a capacidade de aumentar a quantidade de sensores e dados processados à medida que a empresa expande suas operações, mantendo a performance e a confiabilidade do ambiente.

5. A Solução

A solução proposta envolve o desenvolvimento de um sistema de gerenciamento integrado, baseado em uma plataforma de CRM (Customer Relationship Management), que serve como núcleo central para o controle,

observação e administração de sensores, microcontroladores, edge nodes e pods. Essa plataforma é projetada para atender às demandas de um ambiente de *Fog Computing*, proporcionando aos usuários não apenas a capacidade de cadastrar e configurar dispositivos, mas também de monitorar suas condições operacionais e status em tempo real, por meio de um modelo robusto de observabilidade e gerenciamento.

A arquitetura do sistema incorpora elementos fundamentais de infraestrutura distribuída. A observação de pods e containers é detalhada para garantir um nível elevado de transparência sobre o estado dos recursos computacionais. No caso dos pods, aspectos como nome, namespace, status e a versão do Kubernetes são monitorados, enquanto informações relacionadas ao uso de recursos, como CPU, memória e armazenamento, também são registradas. Para os containers, atributos específicos como o nome, imagem utilizada, status operacional e as especificações de recursos alocados são destacados. Além disso, os logs de cada container são armazenados em coleções exclusivas no MongoDB, promovendo uma estrutura organizada e acessível para auditorias e depuração.

Os Edge Nodes, representados por dispositivos como Raspberry Pis, e os microcontroladores desempenham papéis críticos no modelo de computação em névoa. Informações sobre a conectividade, como endereços IP, portas e protocolos de comunicação, são tratadas como elementos essenciais para garantir a estabilidade e a continuidade do monitoramento. Adicionalmente, os microcontroladores incluem mecanismos de telemetria que permitem o acompanhamento em tempo real do uso de CPU, memória e armazenamento, bem como a avaliação do status da rede. Cada microcontrolador é equipado com um daemon responsável por monitorar continuamente as portas físicas e gerenciar a configuração de novos sensores. Sempre que um sensor é conectado, um evento é disparado em uma fila de dispositivos, permitindo que o sistema valide sua presença no banco de dados. Caso o dispositivo não esteja registrado, ele é rejeitado; caso contrário, é configurado automaticamente e passa a ser monitorado pela plataforma.

Os sensores, como dispositivos de entrada de dados no sistema, possuem atributos detalhados que refletem suas características técnicas e operacionais. Entre os elementos destacados estão a precisão, o formato de medição, os intervalos de medida, os limites de temperatura e as informações sobre calibração e manutenção. Esses atributos garantem que os dados coletados sejam confiáveis e que os sensores operem dentro de seus parâmetros ideais. Informações adicionais, como conectividade, modelo, fabricante e localização, também são incorporadas, oferecendo uma visão completa para rastreamento e gestão.

Além disso, o sistema é projetado para gerenciar eventos críticos por meio de um módulo de alertas. Este módulo permite a configuração de mensagens personalizadas, níveis de prioridade e destinatários específicos para notificações via e-mail. Os alertas são disparados com base em eventos detectados pelo sistema, utilizando os dados enviados pelos sensores, a telemetria dos microcontroladores e os logs dos containers. Essa funcionalidade é essencial para permitir respostas rápidas e eficazes a falhas ou condições operacionais críticas.

Do ponto de vista da escalabilidade e integração, a solução foi projetada para suportar a expansão dinâmica. Novos sensores, pods e dispositivos de borda podem ser incorporados sem interrupções no funcionamento do sistema. A integração com filas de eventos assegura a consistência e a segurança na configuração e no registro de novos dispositivos. A separação dos logs por container no MongoDB fornece uma base robusta para auditorias e análises detalhadas, facilitando a identificação de problemas e a tomada de decisões.

A plataforma, portanto, alia flexibilidade e eficiência operacional em um ambiente gerenciável e observável, atendendo às demandas de sistemas distribuídos de alto desempenho, enquanto se mantém economicamente viável e escalável.

O desenvolvimento de um ecossistema robusto e eficiente, como o proposto, enfrenta diversos desafios que exigem uma abordagem holística e cuidadosa. Pensar em uma solução que integre múltiplas camadas de tecnologia, desde sensores e microcontroladores até a infraestrutura em Kubernetes e a integração com a nuvem, demanda um planejamento rigoroso que vai além de aspectos técnicos, abrangendo também estratégias organizacionais e de equipe.

Um dos principais desafios é a definição de padrões de código e design que garantam a criação de uma solução de alto nível, confiável e sustentável. Adotar padrões como *Clean Architecture*, princípios SOLID, e *Domain-Driven Design* auxilia na criação de um sistema modular e fácil de manter, reduzindo a ocorrência de bugs e facilitando futuras expansões. Essa padronização não apenas proporciona uniformidade no desenvolvimento, mas também permite que diferentes membros do time colaborem de forma eficiente, independentemente de sua familiaridade com partes específicas do sistema.

A abstração correta das entidades e seus atributos é outro desafio essencial. Definir claramente como os sensores, microcontroladores,

gateways e serviços interagem entre si, além de estabelecer contratos bem definidos para comunicação entre essas entidades, é fundamental para evitar inconsistências e garantir um fluxo de dados contínuo e confiável. Além disso, essa abstração deve ser flexível o suficiente para permitir a adição de novos dispositivos ou tecnologias sem reestruturar todo o sistema.

Outro aspecto crucial é a criação de uma plataforma gerenciável e observável. Isso significa implementar ferramentas e interfaces que permitam não apenas monitorar o funcionamento do ecossistema, mas também realizar configurações de forma intuitiva e segura. A capacidade de adicionar novos sensores, configurar dispositivos para *Fog Computing* e ajustar parâmetros de processamento deve ser uma tarefa simplificada, possibilitando que a solução evolua sem a necessidade de intervenções complexas.

A escalabilidade também é um ponto de destaque. Projetar um sistema que seja realmente escalável envolve não apenas permitir o crescimento horizontal (com a adição de mais sensores e dispositivos), mas também a otimização de recursos para manter os custos baixos. Isso exige decisões criteriosas na escolha das tecnologias, como o uso de Kubernetes para orquestração e ferramentas leves como microK8S para ambientes de borda. Além disso, é necessário adotar práticas de otimização, como batching e compressão de dados, para reduzir o consumo de largura de banda e armazenamento.

Por fim, o custo da solução deve ser cuidadosamente gerenciado. Criar um ecossistema barato, mas eficiente, exige um equilíbrio entre o uso de tecnologias de ponta e alternativas acessíveis. O uso de microcontroladores de baixo custo e ferramentas open-source, sempre que possível, contribui para a redução dos gastos, sem comprometer a qualidade ou a funcionalidade.

O desafio, portanto, reside em criar uma solução que seja não apenas funcional, mas também gerenciável, observável, escalável e acessível. Pensar de forma holística, integrando padrões de desenvolvimento, abstrações bem definidas e tecnologias flexíveis, é essencial para alcançar um sistema robusto e preparado para as demandas da indústria moderna.

6. Desenvolvimento

O desenvolvimento da solução foi fundamentado na escolha criteriosa de tecnologias e práticas que atendem às demandas de confiabilidade, eficiência e escalabilidade do sistema. A utilização de Python como

linguagem principal para a programação dos microcontroladores e dos nodes de *Fog Computing* permitiu flexibilidade no desenvolvimento de funcionalidades críticas. Paralelamente, o TypeScript foi adotado para o frontend, garantindo maior robustez no CRM, ao mesmo tempo em que promoveu uma experiência de usuário intuitiva e responsiva.

A comunicação em tempo real é facilitada por meio de WebSockets, que permitem a transmissão contínua e bidirecional de dados entre os dispositivos e a plataforma. Essa abordagem assegura que os dados dos sensores, microcontroladores e containers sejam apresentados em tempo real, favorecendo tanto o monitoramento imediato quanto a rápida detecção de anomalias.

A arquitetura do sistema também integra o Kafka como um componente essencial para a mensageria. Ele é utilizado para gerenciar eventos como conexões de novos dispositivos, mensagens enviadas por scanners e sensores, além da troca de informações críticas entre componentes distribuídos. Essa escolha garante alta performance e escalabilidade no processamento de eventos, reduzindo a latência e aumentando a confiabilidade.

Um dos desafios enfrentados foi o envio eficiente de dados dos sensores. Para resolver isso, foram implementadas máquinas de estado que regulam a transmissão em lotes. Esse mecanismo permite o envio de uma quantidade predeterminada de itens, evitando o envio de dados individuais, o que resulta em economia de banda e maior eficiência operacional. Essa abordagem também minimiza os custos operacionais, especialmente em ambientes com recursos de conectividade limitados.

No aspecto de segurança, foi adotado o algoritmo AES-256 combinado com SHA-256 para criptografia de dados em trânsito e em repouso. Essa escolha assegura altos níveis de proteção contra interceptação e acesso não autorizado, alinhando-se às melhores práticas de segurança no desenvolvimento de sistemas distribuídos.

A visualização de dados é um ponto central da solução. O CRM, desenvolvido com TypeScript, serve como uma interface facilitadora para que os usuários possam gerenciar dispositivos, acompanhar medições dos sensores e monitorar os recursos computacionais. Para desenvolvedores e técnicos, ferramentas adicionais, como dashboards personalizados e acesso detalhado a logs, estão disponíveis, garantindo uma experiência holística e ajustada às necessidades de cada público.

A observabilidade e telemetria são alcançadas por meio da coleta e monitoramento contínuo de métricas dos componentes do sistema, como uso de CPU, memória e status de conectividade dos microcontroladores e edge nodes. Essas informações são fundamentais para prever falhas, otimizar o desempenho e tomar decisões embasadas em dados. Cada container tem seus logs armazenados em coleções separadas no MongoDB, o que facilita análises e auditorias detalhadas.

Por fim, o desenvolvimento foi acompanhado por um conjunto abrangente de testes automatizados e manuais. Testes unitários foram aplicados para validar as funções individuais do sistema, enquanto testes de integração e de desempenho garantiram que os componentes operassem em harmonia e sob alta carga. Essa abordagem reduz significativamente o risco de bugs e problemas em produção, assegurando a confiabilidade do sistema desde sua concepção até a entrega final.

7. Discussão

A escolha entre uma arquitetura baseada em Controladores Lógicos Programáveis (CLPs) e outra utilizando containers em um ambiente orquestrado como o Kubernetes envolve uma série de trade-offs técnicos e operacionais. Ambas as abordagens possuem vantagens específicas, sendo mais adequadas para diferentes contextos e demandas.

Os CLPs têm sido amplamente utilizados em ambientes industriais devido à sua confiabilidade, robustez e capacidade de operar em condições adversas. Eles são projetados para executar tarefas específicas de forma determinística, o que os torna ideais para aplicações em que a previsibilidade e a baixa latência são críticas. Contudo, essa abordagem apresenta limitações consideráveis quando o sistema precisa escalar ou incorporar novos dispositivos e funcionalidades. Os CLPs dependem de hardware proprietário, possuem integração limitada com sistemas modernos baseados em nuvem ou borda e oferecem menor flexibilidade para adaptação de novas demandas.

Por outro lado, uma arquitetura baseada em containers, como a implementada em Kubernetes, apresenta uma série de vantagens em cenários de alta dinamicidade e escalabilidade. Containers permitem o desacoplamento entre hardware e software, tornando o ambiente altamente portátil e facilitando a integração com diferentes plataformas. Com o Kubernetes, é possível gerenciar recursos de forma eficiente, alocar automaticamente cargas de trabalho e escalar serviços conforme a necessidade. Além disso, o uso de *Fog Computing* distribui o processamento

de dados, reduzindo a latência e a dependência de conexões constantes com a nuvem.

Outro benefício significativo dos containers é a flexibilidade. Eles possibilitam a execução de aplicações heterogêneas em um mesmo ambiente, atendendo a demandas específicas de sensores, microcontroladores e sistemas de monitoramento. Essa flexibilidade, no entanto, exige maior expertise técnica para configurar e manter a infraestrutura, o que pode representar um desafio inicial para equipes menos experientes. Ainda assim, uma vez implementada, a arquitetura de containers oferece vantagens significativas em termos de observabilidade, telemetria e automação.

Por outro lado, desvantagens como maior complexidade na configuração inicial e a necessidade de recursos computacionais mais robustos devem ser consideradas. O custo de entrada para configurar um ambiente Kubernetes pode ser mais elevado, especialmente quando comparado a sistemas que utilizam CLPs. No entanto, a capacidade de escalar horizontalmente e integrar novas tecnologias rapidamente justifica o investimento em muitos casos.

Embora os CLPs sejam soluções tradicionais e confiáveis, a arquitetura baseada em containers apresenta vantagens notáveis em termos de escalabilidade, eficiência e modernidade, posicionando-se como uma solução superior em projetos que envolvem grandes volumes de dispositivos e demandas complexas de monitoramento.

8. Conclusão

A implementação de um ambiente Kubernetes para monitoramento de sensores representa uma solução escalável, eficiente e alinhada às exigências tecnológicas contemporâneas. Diferentemente das arquiteturas tradicionais baseadas em CLPs, que oferecem confiabilidade em cenários específicos, o uso de containers traz uma abordagem moderna, flexível e capaz de atender a uma diversidade de demandas.

Por meio da combinação entre Kubernetes, *Fog Computing* e boas práticas de design, a solução desenvolvida permite a gestão eficiente de sensores, microcontroladores e dispositivos de computação de borda, promovendo uma integração fluida entre componentes. Essa arquitetura não apenas facilita o escalonamento horizontal, como também garante observabilidade e telemetria detalhadas, fundamentais para a tomada de decisões em tempo real. Apesar dos desafios iniciais relacionados à configuração e ao gerenciamento, a adoção de containers se destaca como a

escolha ideal para atender a cenários complexos, garantindo eficiência operacional e suporte a inovações tecnológicas.

Assim, a implementação descrita neste trabalho enfatiza a viabilidade e as vantagens de arquiteturas modernas e distribuídas, demonstrando como o Kubernetes pode transformar ambientes de monitoramento, promovendo um ecossistema robusto, sustentável e adaptável ao futuro.

9. Referências

JUNIOR C., EUELITON M; QUINTINO, LUIS FERNANDO; FILHO DUARTE, PEDRO. **Comunicação entre CLP - IHM - Excel por meio de protocolo OPC DA, estudo de caso**. UFABC - São Bernardo do Campo - SP 2016. Disponível em <https://www.researchgate.net/profile/Alexandre-Andrade-3/publication/308919805_Comunicacao_entre_CLP_-_IHM_-_Excel_por_meio_de_protocolo_OPC_DA_estudo_de_caso/links/5ea0c464299bf143893ff7f2/Comunicacao-entre-CLP-IHM-Excel-por-meio-de-protocolo-OPC-DA-estudo-de-caso.pdf>. Acesso em 13 de nov 2024.

DOCKER INC. **USE CONTAINERS TO BUILD, SHARE AND RUN YOUR APPLICATIONS**. Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em: 18 de nov 2024.

DOCKER INC. **WHAT IS DOCKER?**. Disponível em: <https://docs.docker.com/get-started/docker-overview/>. Acesso em: 18 nov 2024.

GOOGLE INC. **O QUE É KUBERNETES?**. Disponível em: <https://cloud.google.com/learn/what-is-kubernetes?hl=pt-BR>. Acesso em 18 nov 2024.

MORAIS L. DA SILVA. **Utilização de armazenamento definido por software em uma arquitetura hiperconvergente de containers**. UNB - Brasília 2019. Disponível em <http://www.realp.unb.br/jspui/handle/10482/36759>. Acesso em 13 de nov 2024.

SILVEIRA, LEONARDO; Q. LIMA, WELDSON. **Um breve histórico conceitual da Automação Industrial e Redes para Automação Industrial. UFRN-PPgEE. Rio Grande do Norte**. 2003. Disponível em: <https://d1wqtxts1xzle7.cloudfront.net/43839581/Redes_Industriais_e_Automacao-libre.pdf?1458266012=&response-content-disposition=inline%3B+filena

me%3DUm_breve_historico_conceitual_da_Automac.pdf&Expires=1730332340&Signature=dpYqf-h8WlD7R268cKnDDadxbGq9N9ViSujhB-sLMSjM0zrnPBb6H0L4wVqWkb1QhsmojYNyPuyxc54VXzU0vjij248ppJiarcH1ZiXJdXBB5ybsYpaYNJhb46MAiaJ6NPwCK72MMY1PKNHRFBvn81P6ft2z7MyJaN4g-ghExBHb8OUBbldX51-EZiqtf6aceSM6PO5knnep-z96PdZVQVDA5nOaHlwwX8jvTvC3n4tgDuKsQZOe2-OTz7khYY0KxaiIMa39jTclAUNFBnfjVWzuis7mtpmKXOYipAzwEhRWZbuCvYQrpZg4Rnm~5-91K2W0nDRUtHD8HYOPDwv36A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA>. Acesso em: 30 out. 2024.

OLIVEIRA, JOÃO PEDRO B. **AUTOMAÇÃO DE CÉLULA PARA PREPARAÇÃO DE AMOSTRAS DE MINÉRIO DE FERRO PARA ANÁLISES LABORATORIAIS UTILIZANDO CLP E ROBÔ INDUSTRIAL**. **Escola de Minas - UFOP**. 2024. Disponível em: <https://monografias.ufop.br/bitstream/35400000/7018/3/MONOGRAFIA_AutomaçãoCélulaPreparação.pdf> . Acesso em: 06 de nov 2024.

SIEMENS. **CLP SIMATIC S7 1200**. 2024. Disponível em: <https://www.siemens.com/br/pt/produtos/automacao/controladores/s7-1200.html>. Acesso em: 06 de nov 2024.

CARVALHO DE, DIEGO B. **IMPORTÂNCIA DA IMPLANTAÇÃO DE CONTROLADOR LÓGICO PROGRAMÁVEL PARA A AUTOMAÇÃO INDUSTRIAL**. 2017. Faculdade Pitágoras, Poços de Caldas. Disponível em: <https://repositorio.pgsscogna.com.br/bitstream/123456789/19626/1/DIEGO%20BERALDO%20DE%20CARVALHO.pdf> . Acesso em: 06 de nov 2024

MONOSTORI, L., et al**. CYBER-PHYSICAL SYSTEMS IN MANUFACTURING.** *CIRP Annals*, vol. 65, no. 2, 2016, pp. 621–641. Disponível em: <https://www.researchgate.net/profile/Botond_Kadar/publication/306426761_Cyber-physical_systems_in_manufacturing/links/5b7275b992851ca650583cf9/Cyber-physical-systems-in-manufacturing.pdf?__cf_chl_tk=qsPgf4OZGvADezAYOjW7_vrPufUvaGgm.7LLVXR0t8g-1731961873-1.0.1.1-byRw8xI7iA7sX.xA0Z_gUAqqoxLLC0UmKuhG2nqkKVY>. Acesso em 18 de nov 2024.

MAHMOOD, Z., et al. **FOG COMPUTING: CONCEPTS, PRINCIPLES AND RELATED TECHNOLOGIES**

AMAZON INC. **O QUE É A COMPUTAÇÃO EM NUVEM?**. Amazon Web Services. Disponível em: <https://aws.amazon.com/pt/what-is-cloud-computing/>. Acesso em 18 de nov 2024.

MARTIN, ROBERT C. **CLEAN ARCHITECTURE**. 2018. p-195. Disponível em <https://github.com/ropalma/ICMC-USP/blob/master/Book%20-%20Clean%20Architecture%20-%20Robert%20Cecil%20Martin.pdf>. Acesso em 19 de nov 2024.

TRES, TACIANO; FURTADO V., OLINTO J. **UTILIZANDO PADRÕES DE PROJETO NO DESENVOLVIMENTO DE APLICAÇÕES TEMPO REAL.** p-7. Departamento de Informática e Estatística - Centro Tecnológico Universidade

Federal de Santa Catarina 2019. Disponível em <https://www.researchgate.net/profile/Olinto-Furtado/publication/343125048_Utilizando_Design_Patterns_no_Desenvolvimento_de_Aplicacoes_Tempo_Real/links/5f1e3bc792851cd5fa4b153b/Utilizando-Design-Patterns-no-Desenvolvimento-de-Aplicacoes-Tempo-Real.pdf>. Acesso em 19 de nov 2024.