Universidade Federal de Ouro Preto - UFOP - Escola de Minas - Colegiado do curso de Engenharia de Controle e Automação - CECAU

Glauco de Oliveira Leme

DESENVOLVIMENTO DE PRODUTO MINIMAMENTE VIÁVEL PARA SISTEMAS INDUSTRIAIS DE AUTOMAÇÃO

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, 2018

Glauco de Oliveira Leme

DESENVOLVIMENTO DE PRODUTO MINIMAMENTE VIÁVEL PARA SISTEMAS INDUSTRIAIS DE AUTOMAÇÃO

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: prof.ª Adrielle de Carvalho Santana, Msc.

Ouro Preto, 2018

Glauco de Oliveira Leme

DESENVOLVIMENTO DE PRODUTO MINIMAMENTE VIÁVEL PARA SISTEMAS INDUSTRIAIS DE AUTOMAÇÃO/ Glauco de Oliveira Leme. – Ouro Preto, 2018-

55 p. : il. (algumas color.) ; 30 cm.

Orientador: prof.ª Adrielle de Carvalho Santana, Msc.

Monografia de Graduação em Engenharia de Controle e Automação – Universidade Federal de Ouro Preto - UFOP - Escola de Minas - Colegiado do curso de Engenharia de Controle e Automação - CECAU, 2018.

1. Micro-serviços. 2. Indústria 4.0. I. Adrielle de Carvalho Santana. II. Universidade Federal de Ouro Preto. III. Colegiado do curso de Engenharia de Controle e Automação. IV. Desenvolvimento de Produto Minimamente Viável para Sistemas Industriais de Automação.

CDU 02:141:005.7

ATA DE DEFESA

Ao 11º dia do mês de outubro de 2018, às 10:00h, na Escola de Minas - UFOP, reuniu-se a Comissão Avaliadora designada para julgar a Monografia do graduando **GLAUCO DE OLIVEIRA LEME** de Engenharia de Controle e Automação intitulada "**DESENVOLVIMENTO DE PRODUTO MINIMAMENTE VIÁVEL PARA SISTEMAS INDUSTRIAIS DE AUTOMAÇÃO**", sob orientação da Profa. Adrielle de Carvalho Santana, sendo a referida Comissão composta pelos professores Adrielle de Carvalho Santana, Agnaldo José da Rocha Reis, Vinícius Marinho Silva. A Comissão Avaliadora resolveu considerar o trabalho APROVADO por ter atendido as exigências para defesa e recomendações da banca examinadora.

Ouro Preto, 11 de outubro de 2018.

Profa. M. Sc. Adrielle de Carvalho Santana – Orientadora

Prof. Dr. Agnaldo José da Rocha Reis – Professor Convidado

Prof. Vinícius Marinho Silva – Professor Convidado

GLAUCO DE OLIVEIRA LEME – Aluno

Dedico este trabalho à Mercês, Cideonir e a Pamela, companhias fiéis durante minha jornada.

Agradecimentos

Agradeço primeiramente à minha família que exerceu papel fundamental em minha formação.

Agradeço a minha namorada Pamela por todo incentivo e companheirismo, sempre que precisei.

Agradeço também aos meus colegas de curso e aos professores pelo suporte e aprendizado ao longo do curso. Especialmente ao Prof. Agnaldo José Reis e a Prof.ª Adrielle Carvalho de Santana.

À todos os aqueles responsáveis, direta e indiretamente, pelo meu processo de educação e formação acadêmica.

'As pessoas mais criativas são motivadas pelos maiores dos problemas que se apresentam diante delas" (Dr. Neil deGrasse Tyson)

Resumo

O uso de softwares em código aberto proporciona à indústria uma oportunidade de baixo custo de investimento para inserção de novas tecnologias para acompanhar a crescente competição e inovações de mercado. A maneira como essas tecnologias serão empregadas e absorvidas que irá agir como fator determinante do sucesso neste processo de evolução à Indústria 4.0. Portanto é necessário estudar as estratégias, arquiteturas e modo de desenvolvimento de tais ferramentas dentro de um ambiente industrial, que requer, acima de tudo, ganho no processo acompanhado de estabilidade e otimização de recursos. Neste trabalho será apresentado estudo e análise de técnicas e ferramentas que envolvem o desenvolvimento de software a fim de apresentar uma arquitetura de um produto minimamente viável que possa ser inserido em ambiente industrial com mínima influência destrutiva. Também será demonstrado um estudo de caso de forno rotativo, cuja função é o aquecimento de blocos de aço em etapa precedente ao processo de laminação. O emprego da arquitetura proposta possibilitou a inclusão de modelagem matemática da temperatura dos blocos para análise dos dados dos blocos enfornados de maneira a gerar ganhos consideráveis ao processo de produção.

Palavras-chaves: Micro-serviços, REST, MVP, Software, Indústria.

Abstract

The use of open source software allows the industry a low-cost investment opportunity to incorporate new technologies and keep pace with growing competition and market innovations. How these technologies will be employed and absorbed will act as a determinant fact of success in the process of evolution to the Industry 4.0. Therefore, it is necessary to study the strategies, architectures and development of such tools within an industrial environment, which is mandatory to generate gain in the process followed by stability and optimization of resources. At this undergraduate thesis, a study and analysis of techniques and tools about software development will be shown in order to present an architecture of an minimal viable product with minimal destructive impact. Will also be shown a case study of an rotary hearth furnace, which task is to heat the steel billets in a imediataly preceding stage than rolling. The usage of the proposed architecture allowed the integration of a mathematical modelling of the billets temperature for the data analysis in a way to generate several benefits to the production proccess.

Key-words: Microservices, REST, MVP, Software, Industry.

Lista de ilustrações

Figura 1 Grupo baseado em camadas de tecnologia 32
Figura 2 Grupo baseado em regras de negócio 32
Figura 3 Banco de dados voltados as aplicações 34
Figura 4 Fluxo de informação do RabbitMQ 36

Figura 5 Representação das seções de um forno de soleira rotativa 41
Figura 6 Forno de soleira rotativa 42
Figura 7 Mapa de aquecimento dos blocos ao longo do forno 43
Figura 8 Visão geral da arquitetura proposta 46

Lista de tabelas

Tabela 1 Exemplos de requisições REST 29
Tabela 2 Comparação entre dados relacionais e não relacionais 37

Lista de abreviaturas e siglas

MVP	Produto minimamente viável (*Minimum Viable Product*)
REST	Estado de representação de estados (*Representational State Transfer*)
Web	Sistema hipertextual que opera pela internet
AMQP	Protocolo avançado de fila de mensagens (*Advanced Message Queuing Protocol*)
STOMP	Protocolo de transmissão de mensagens orientado em textos(*Streaming Text Oriented Messaging Protocol*)
XML	Linguagem de marcação extensível (*Extensible Markup Language*)
JSON	Objeto de notação em JavaScript (*JavaScript Object Notation*)

Sumário

1 INTRODUÇÃO 23
1.1 Contextualização 23
1.2 Justificativa 24
1.3 Objetivo 24
1.4 Organização e estrutura 25

2 REVISÃO TEÓRICA 27
2.1 REST 27
2.2 MICROSERVIÇOS 30
2.3 SOFTWARES 34
2.3.1 FILA DE MENSAGENS 34
2.3.2 BANCO DE DADOS 35
2.3.3 LINGUAGEM DE PROGRAMAÇÃO 38

3 DESENVOLVIMENTO 41
3.1 Estudo de caso - Forno Rotativo 41
3.2 Arquitetura proposta 45
3.2.1 Sistema operacional 45
3.2.2 Fila de mensagens 47
3.2.3 Aplicação do modelo matemático 47
3.2.4 Cliente para exibição 48

4 DISCUSSÃO 49

Conclusão 51

Referências 53

1 INTRODUÇÃO

1.1 Contextualização

Desde a primeira revolução no final do século 19, a indústria siderúrgica é utilizada como um forte indicador de progresso econômico de um país. Os produtos oriundos de ligas de ferro e aço tem grande importância no desenvolvimento infraestrutural e econômico. O aço possui propriedades de grande maleabilidade, boa resistência, durabilidade, elasticidade e excelente condutividade térmica. Por isso, possui empregabilidade em diversos setores, tais como construção civil, transporte, energia e é utilizado até mesmo em produtos domésticos.

Durante a segunda revolução industrial, ao longo do século 20, houve inúmeras melhorias no processo de fabricação do aço, sejam elas do ponto de vista operacional e tecnológico. Melhorias fomentadas pelas necessidades do mercado, competitividade, ou visando atender critérios de qualidade dos clientes, exigências trabalhistas ou simplesmente critérios econômicos, ao reduzir os custos operacionais e otimizar a utilização de recursos. No final do século 20, deu-se início à terceira revolução industrial. Com o avanço tecnológico e diminuição de custos relacionados à maquinário e mão de obra, investimentos em infraestrutura, eletrônica e automação, proporcionaram à indústria siderúrgica a utilização de ferramentas que agregam mais qualidade ao produto final e que, ainda que demandam maior custo operacional, geram economia ao longo prazo por trazer um fluxo de produção mais otimizado.

A automação industrial, portanto, teve impacto desde aos procedimentos de chão de fábrica até o nível gerencial. Esteve presente na substituição de comandos responsivos ou situações periculosas com o emprego de equipamentos mecanizados que fazem uso de comandos de programação lógica, aumentando segurança e ritmo de produção. Além da abrangente coleta de dados para que seja possível profunda análise dos procedimentos operacionais, utilização de manutenção preventiva para reduzir custos e no auxílio ou substituição da tomada de decisão das atividades humanas através do uso de sistemas de supervisão e controle.

Já o início do século 21, a indústria é marcada pela chegada da quarta revolução industrial, a denominada indústria 4.0. Segundo Lasi et al. (2014), existem conceitos fundamentais os quais definem a Indústria 4.0 dos quais é possível destacar:

- Fábrica inteligente: manufatura deve ser completamente equipada com sensores,

atuadores e sistemas autônomos;

- Sistemas ciber-físicos: os equipamentos possuem parametrização total e são compostos de elementos computacionais e embarcados colaborativos de maneira que não é possível obter o estado atual sem avaliar também os parâmetros de processo de origem digital;
- Auto-organização: quebra na hierarquia de processos faz com que os sistemas hajam de maneira descentralizada;
- Integração de sistemas: a informação é capaz de sair e chegar até onde é necessário de maneira automática;
- Adaptações às necessidades humanas: inversão da lógica em que os operários se adaptam para realizar as atividades de produção;
- Predomínio de responsabilidades sociais: sustentabilidade e otimização no uso de recursos estão no foco do processo de produção.

Tais conceitos, uma vez distantes da realidade, nos apresentam a um grande desafio ao propor desenvolvimento de atividades com alto grau de complexidade, dinâmicas e que exigem integração entre todo o sistemas de produção. Grandes mudanças, apesar de envolverem maior foco em áreas de automação e de tecnologia da informação, requerem um grande planejamento e investimento para construção de uma nova arquitetura que dê suporte a longo prazo.

1.2 Justificativa

Há uma grande e crescente demanda da indústria para que haja inclusão de novos métodos e ferramentas de software para satisfazer novas necessidades do mercado e adaptação à era da indústria 4.0. Porém é importante ressaltar que não é viável, e em alguns casos até mesmo impossível, desfazer do que já está desenvolvido para a criação de um novo sistema que seja amigável à inclusão de tais ferramentas e recursos. Para que seja viável é importante adicionar e não subtrair. O desafio consiste portanto no desenvolvimento de ferramentas que estejam adaptadas às diretrizes da indústria 4.0 e que apresentem ganho real ao processo sem interferência destrutiva nos processos atuais e já considerados estáveis a nível de produção.

1.3 Objetivo

Visto essa necessidade da indústria em se adaptar à Indústria 4.0, este trabalho consiste no estudo e análise de ferramentas para apresentar uma arquitetura para construção

e utilização de um produto minimamente viável (MVP) desenvolvida para um caso real da indústria (forno de soleira rotativa) e que atenda aos requisitos de tal adaptação. Para tal, este produto deve ser capaz de:

- Comunicar com outros sistemas que estão em operação;
- Estabelecer métricas para inclusão de novas funcionalidades relacionadas a sistemas;
- Disponibilizar aos usuários interface para informações relevantes do processo de produção;
- Fazer uso de sistemas *open source*.

1.4 Organização e estrutura

O restante do trabalho está organizado em quatro etapas, das quais consistem em revisão de literatura, estudo de caso, desenvolvimento e considerações finais. Na revisão serão apresentados quais os fundamentos e softwares que poderão compor a solução final. No estudo de caso é apresentado um problema real da indústria e como as soluções padrões de automação não fornecem atualmente uma solução otimizada e viável. No desenvolvimento é apresentada a arquitetura proposta para o *MVP* e no capítulo final serão abordadas as considerações finais sobre esta solução.

2 REVISÃO TEÓRICA

2.1 REST

Em um ambiente corporativo, a utilização de sistemas computacionais é extremamente comum, seja para sistemas voltados à administração, recursos humanos ou à manufatura. Tais sistemas normalmente operam de maneira independente e isolados. A possibilidade de haver intercomunicação entre tais sistemas ou até mesmo a exposição de serviços específicos destes sistemas para acesso sistemas externos à rede, se tornou uma necessidade para maior integração de informação e serviços computacionais. Para saciar essa necessidade, se faz necessária a adoção de uma arquitetura de comunicação funcional e robusta, preferencialmente de simples desenvolvimento e de baixa complexidade.

O REST é um estilo de arquitetura de comunicação para desenvolvimento de serviços Web, que obteve popularidade devido à sua simplicidade. Sua arquitetura incorpora o uso de recursos já existentes, como exemplo, um destes recursos é o protocolo HTTP (*Hypertext Transfer Protocol*), que constitui a base da comunicação Web e é amplamente utilizado pela comunidade dos desenvolvedores de serviços web (WEBBER; PARASTATIDIS; ROBINSON, 2010).

A maneira como o REST incorpora estes recursos implica em uma relação de interdependência com sua arquitetura. Para representação dos estados de comunicação são utilizados os padrões do já citado protocolo HTTP. Já para criptografia, são adotados os padrões SSL (*Secure Sockets Layer*) e TLS (*Transport Layer Security*), também amplamente utilizados por desenvolvedores de serviços Web (GROUP et al., 1999).

Ao adotar a arquitetura REST, é possível planejar a construção dos serviços com foco nas requisições e problemática da solução desejada. A grosso modo, desde que a solução planejada seja capaz de realizar e escutar requisições HTTP, o REST se torna aplicável. Tais definições como linguagem de programação, sistema operacional e interface de dados podem ser analisadas livremente das limitações da arquitetura. Isso é uma vantagem muito importante para a arquitetura REST, ter independência destes fatores a torna aplicável a diversos tipos de solução, o que justifica e intensifica seu uso. Em contramão, as limitações dos padrões adotados também apresentam ligação direta com as limitações da arquitetura REST. Desvantagens do protocolo HTTP, dos padrões de criptografia e segurança, portanto, também se fazem presentes.

Em outras arquiteturas, como o SOAP(*Simple Object Access Protocol*), há certas

limitações como a necessidade do uso do XML como formato utilizado, o que dificulta o aprendizado e limita a equipe de desenvolvimento. Há também a necessidade de inclusão de cabeçalho junto com os dados o que aumenta o volume de dados trafegados. Em compensação é possível aplicar protocolos mais robustos de segurança afetando a maneira como o cliente acessa os recursos do servidor. Outra clara desvantagem da arquitetura REST é ausência do tratamento em caso de problemas de comunicação. É preciso realizar outras estratégias como uso de sistemas de fila de mensagens ou uma rotina de repetição no cliente para que os problemas sejam contornados.

Segundo Fielding e Taylor (2000) os sistemas baseados em REST devem ter baixa complexidade para facilitar o desenvolvimento e atender as demandas iniciais de maneira mais rápida. Mas com o tempo, as demandas sofrem alteração ou novas situações surgem, e a arquitetura precisa de se comportar de maneira a prover crescimento sem se limitar nos requisitos da solução inicial. Para tal, são atribuídas a arquitetura REST condições de contorno, sendo algumas delas:

- Interface uniforme: Para cada recurso do servidor deve estar exposto aos clientes uma única URL, sendo este recurso manipulável usando apenas os métodos do protocolo HTTP, como *Delete*, *Put* e *Get*;
- Cliente-servidor: Deve haver um esboço claro entre estes dois componentes de maneira que a cada um é incumbido papéis diferentes. Enquanto o cliente é responsável pelas solicitações o servidor controla os dados, regras de negócio e segurança. Desta forma o crescimento dos componentes é independente;
- Cachê de recursos: para redução de latência e melhora do desempenho se faz necessário o armazenamento de dados e recursos em cachê sempre que possível;
- Operações independentes do estado: As iterações entre cliente e servidor devem ocorrer de maneira que o gerenciamento do estado atual deve ocorrer no cliente, não afetando as operações de interface. A requisição deve conter todas as informações necessárias para entendimento da mesma.

Para melhor entendimento do funcionamento da arquitetura REST é importante entender como o HTTP funciona. O HTTP é um protocolo que permite enviar e receber informações, apresentando um conjunto de regras que determinam que tipo de informações podem ser trocadas entre os membros de uma rede. Estes membros são identificados como servidor ou cliente e, por padrão, a requisição é feita sempre pelo cliente. Após a requisição, o papel do servidor é processar os dados para realização da resposta.

As requisições HTTP são feitas a partir de corpo e cabeçalho. O corpo contém informações que a serem transmitidas de acordo com as especificações do cabeçalho. Na

arquitetura REST adere-se grande importância ao conteúdo do cabeçalho, que contém as especificações necessárias à formação de comandos entre os componentes.

Ao realizar requisições REST, a iteração do cliente com o servidor é feita com o uso de simples URLs (*Uniform Resource Locator*) que contém em seu corpo os dados envolvidos na requisição, como mostra a tabela 1. A utilização das instruções que compõem os cabeçalhos se faz presente utilizando os verbos de chamada das URLs (GROUP et al., 1999). Estes verbos podem ser:

- GET: informa ao servidor que está sendo realizada uma operação de requisição de dados apenas para leitura. Não se deve alterar informações através de uma requisição deste tipo;
- PUT: utilizada quando desejamos criar ou atualizar uma informação identificada no corpo da mensagem. O servidor portanto recebe os dados e faz as alterações necessárias;
- DELETE: funciona de maneira semelhante ao PUT, porém com a intensão de remoção dos dados;
- POST: também envia informações para o servidor, muito semelhante ao método PUT, mas para ações mais genéricas, como por exemplo agir de gatilho para uma operação específica do servidor;
- PATCH: aplica modificações parciais em um objeto do servidor. Seria algo muito semelhante aos comandos de atualização de colunas de um banco de dados, a nível de comparação.

Tipo	**URL**	**Descrição**
Get	servidor/users/id	Identificação de um usuário
Get	servidor/users/id/data	Dados relacionados ao usuário
Post	servidor/users/id/data	Salva informações de um usuário
Get	servidor/users	Lista de usuários

Tabela 1: Exemplos de requisições REST

Após envio do comando, o fluxo normal é o servidor realizar a operação e, se necessário, enviar resposta ao cliente. Porém, é possível que ocorram deste erros de comunicação a erros internos no servidor. Portanto, se faz uso de códigos de resposta, também padrões do protocolo HTTP, podendo citar como alguns exemplos:

- 200 - OK: indica que a requisição foi bem sucedida;

- 201 - Criação OK: indica que a requisição PUT ou POST com a criação de um recurso foi bem sucedida;
- 400 - Solicitação inválida: requisição feita de maneira incorreta;
- 404 - Página não encontrada: o que foi requisitado não pode ser encontrado, ligadas a URLs inexistentes;
- 401 - Não autorizado: indica a necessidade de autenticação para acesso ao recurso;
- 500 - Erro interno do servidor: erro muito genérico que indica que houve falha devido a alguma circunstâncias do servidor.

2.2 MICROSERVIÇOS

O conceito de micro serviços consiste em desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um executando em seu próprio processo e comunicando-se com mecanismos leves, geralmente uma API de recurso HTTP. Esses serviços são criados com base nos recursos de negócios e implementados de maneira independente por máquinas de implantação totalmente automatizadas. Há um mínimo de gerenciamento centralizado desses serviços, que pode ser escrito em diferentes linguagens de programação e usar diferentes tecnologias de armazenamento de dados (THÖNES, 2015).

Os conceitos utilizados por micro serviços não é emprega nada revolucionário. Ele aplica diversas teorias que tiveram sucesso em diversos outros arquiteturas, como por exemplo a produção enxuta (*LEAN*) que aborda conceitos como variedade de práticas operacionais, equipes de trabalho, sistemas integrados e fabricação em células. Em micro serviços estes conceitos estão focados ao desenvolvimento de serviços e software (SHAH; WARD, 2003).

Para melhor entendimento do que é a estrutura de micro serviços, se faz necessário um melhor entendimento da estrutura monolítica, ou seja, de uma única unidade. Normalmente os sistemas são construídos em três partes principais: uma interface de usuário do lado do cliente que consiste de páginas HTML e JavaScript, executadas em um navegador na máquina do usuário; um banco de dados que consiste de muitas tabelas inseridas em um gerenciamento de banco de dados comum e comumente relacional; e uma aplicação do lado do servidor.

O serviço no servidor manipulará solicitações do cliente, executará a lógica do domínio, recuperará e atualizará dados do banco de dados e selecionará e preencherá as solicitações do cliente para que sejam enviadas ao navegador. Este serviço do servidor opera de maneira monolítica, ou seja, se forem necessárias alterações no sistema automaticamente é necessária a criação e a implantação de uma nova versão do serviço do lado do servidor.

Esse servidor monolítico é uma maneira natural de abordar a construção de tal sistema. Toda a sua lógica para manipular uma solicitação é executada em um único processo, permitindo que você use os recursos básicos da sua linguagem para dividir a aplicação em classes e funções.

Serviços monolíticos podem ter sucesso, mas há diversas dificuldades em seu uso especialmente à medida que mais aplicativos são implantados na nuvem, o que é uma tendência natural. Não é interessante apresentar uma estrutura travada em mudanças de maneira que os componentes estão interligados, uma pequena alteração requer que todo o serviço seja reconstruído e reimplantado. No macro, se torna difícil garantir que ao longo do tempo, será possível manter uma boa estrutura modular pois o escalonamento requer que todo o sistema escalone, em vez de partes dele que exigem maior recurso.

Este tipo de problema leva à ideia de uma arquitetura que opere como conjuntos de serviços. Além do fato de que os serviços são independentes e escalonáveis, a possibilidade de arquitetura REST para intercomunicação dos serviços permite até mesmo que diferentes serviços sejam escritos em diferentes linguagens de programação e, portanto, possam ser gerenciados por equipes diferentes. (NEWMAN, 2015)

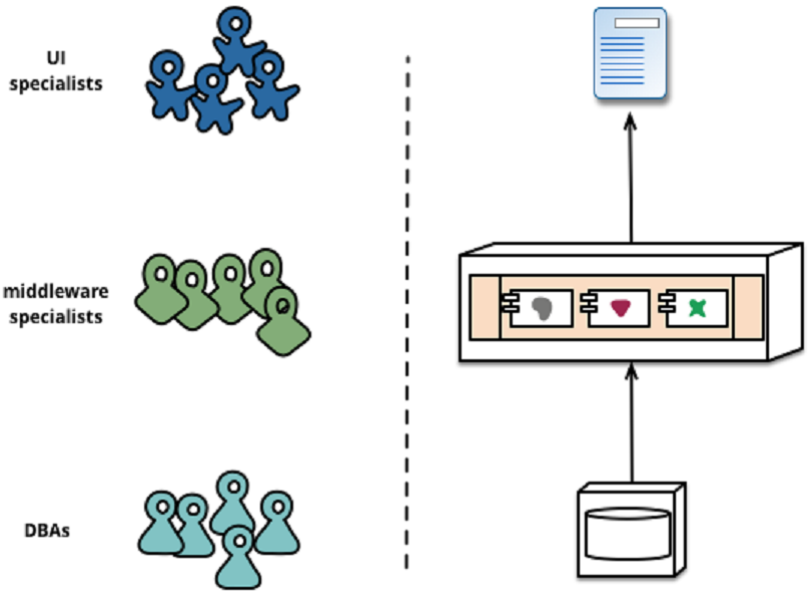
Os micro serviços apresentam diversas características, tais como:

- Criação de componentes através de serviços: apesar do constante uso de bibliotecas, a principal forma de criação de componentes é dividir-se em serviços. A diferença entre biblioteca e serviço é que as bibliotecas são componentes vinculados a um programa e que realizam chamadas de função na memória. Já os os serviços são componentes fora de processo que se comunicam com um mecanismo, por exemplo REST. Um dos principais motivos para usar serviços como componentes é que estes são implementados de maneira independente, ou seja, a alteração de um serviço não implica em alterações em outro serviço, o que não acontece quando há alterações em bibliotecas. Poderão haver mudanças das interfaces de serviço que resultam em alguma atualização, mas o objetivo de uma implementação de micro serviços é minimizar quaisquer impactos deste tipo. Outro aspecto importante é que os serviços facilitam as chamadas a recursos usando mecanismos explícitos de chamada remota (LEWIS; FOWLER, 2014).

- Organização em volta das regras de negócio: ao dividir um projeto normalmente a escolha é o gerenciamento se concentrar na camada de tecnologia, levando a equipes de interface do usuário, equipes lógicas do lado do servidor e equipes de banco de dados, como pode ser visto na figura 1. Porém essa estrutura pode levar a uma demora no processo de desenvolvimento de tarefas muito simples que envolvam mais de uma equipe. A abordagem de micro serviço é diferente, dividindo em torno da capacidade de negócios, como pode ser visto na figura 2. Esses serviços

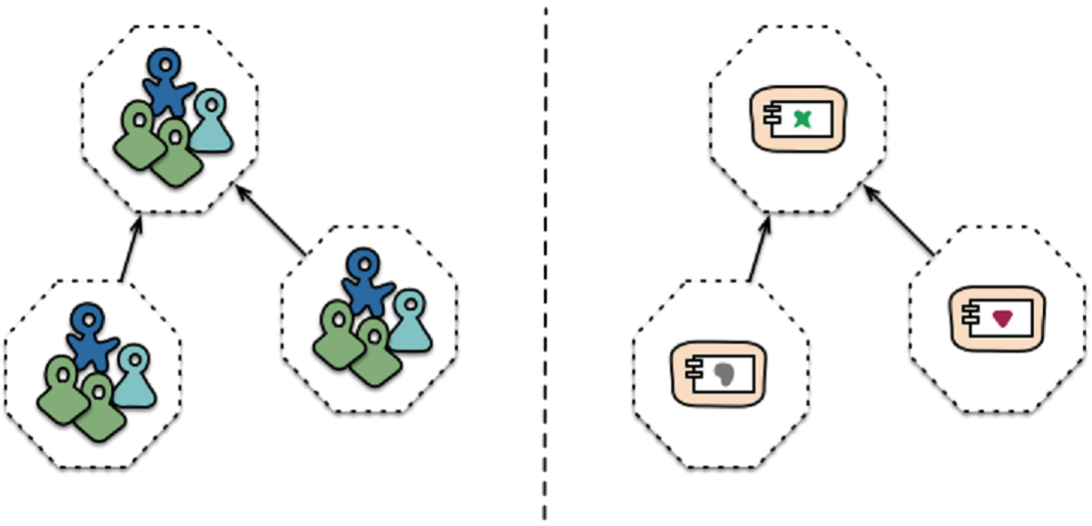
exigem uma implementação de software de grande escala para essa área de negócios, incluindo interface de usuário, armazenamento persistente e até mesmo envolver colaboração externa. Consequentemente, as equipes são multifuncionais, incluindo diversas habilidades necessárias para o desenvolvimento: experiência do usuário, banco de dados e gerenciamento de projetos.

Figura 1: Grupo baseado em camadas de tecnologia



Fonte: (DMITRY; MANFRED, 2014)

Figura 2: Grupo baseado em regras de negócio



Fonte: (DMITRY; MANFRED, 2014)

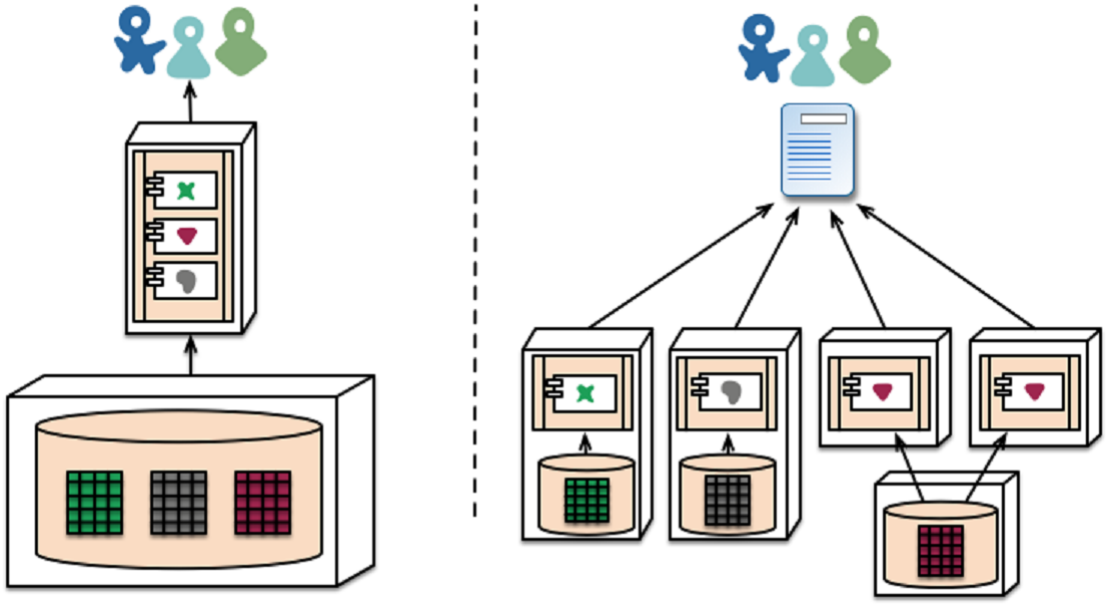
- Produtos ao invés de projetos: A maioria dos esforços de desenvolvimento de aplicativos que vemos usam um modelo onde o objetivo é entregar um software. Após a conclusão, o software é entregue a uma organização de manutenção e a equipe de projeto que a construiu é desfeita. Ao usar micro serviços há uma tendência de evitar esse modelo, preferindo a noção de que uma equipe deve possuir um produto durante toda a sua vida útil. Uma inspiração comum seria onde uma equipe de desenvolvimento assume total responsabilidade pelo software em produção. Isso leva os desenvolvedores ao contato diário do comportamento do software na produção e aumenta o contato com os usuários finais, já que eles precisam assumir parte da carga de suporte do produto final. Em vez de considerar o software como um conjunto de funcionalidades estáticas,há a construção de uma ferramenta de software que pode ajudar seus usuários a aprimorar a capacidade de negócios e facilitar a criação de relacionamentos pessoais entre os desenvolvedores de serviços e seus usuários (DMITRY; MANFRED, 2014).

- Inteligência concentrada nos nós: em um sistema monolítico, os componentes estão sendo executados no processo e a comunicação entre eles é via chamada de método ou chamadas de função. Ao adotar micro serviços, há uma forte tendência há criar nós inteligentes e meios de comunicação menos inteligentes. Isso se torna ainda mais verídico pelo uso da arquitetura REST.

- Organização descentralizada: a descentralização do gerenciamento de dados ocorre ao dividir um domínio complexo em vários contextos limitados e mapeia as relações entre eles.Além de descentralizar decisões sobre modelos conceituais também ocorre a descentralização das decisões de armazenamento de dados. Os micro serviços preferem permitir que cada serviço gerencie seu próprio banco de dados, seja instâncias diferentes da mesma tecnologia de banco de dados ou sistemas de banco de dados totalmente diferentes. Tal abordagem é chamada de Persistência Poliglota e pode ser visto um comparativo na figura 3. É possível o uso da persistência poliglota em sistemas monolíticos mas seu uso é mais frequente em micro serviços.

- Design evolutivo: Colocar componentes em serviços adiciona uma oportunidade para um planejamento de lançamento para utilização em produção mais granular. Com um sistema monolítico, qualquer alteração requer uma compilação e implementação de todo o sistema. Com micro serviços, no entanto, é necessário lançar apenas o que foi modificado. Isso pode simplificar e acelerar o processo de liberação. A desvantagem é que você precisa se preocupar com mudanças em um serviço que esteja influenciando outros serviços que o consumam. A abordagem de integração tradicional é tentar lidar com esse problema usando o controle de versão (BREWER, 2000).

Figura 3: Banco de dados voltados as aplicações



Fonte: (DMITRY; MANFRED, 2014)

2.3 SOFTWARES

A abertura de código de softwares em grandes empresas apresentou uma crescente evolução nos últimos anos. Esta abertura envolveu uma enorme participação de desenvolvedores, de múltiplas empresas e países, que culminou no desenvolvimento de melhorias e novas ferramentas de maneira colaborativa. Como demonstrado por Doukas e Thramboulidis (2011), o uso de softwares denominados FLOSS *Free/Libre/Open Source Software* pode trazer benefícios à indústria pela importação de boas práticas, tais como: redução de débitos técnicos, dar maior liberdade aos métodos de trabalho dos desenvolvedores, definir métricas para incorporação de novos membros, aumento da disponibilidade e visibilidade de informação e documentação.

Como requerimento do projeto é a utilização de softwares de uso gratuito, nesta seção serão listados diversos softwares que podem ser classificados de acordo com a funcionalidade que cada um pode exercer dentro do sistema. Encontra-se listado as opções consideradas para cada tipo de função.

2.3.1 FILA DE MENSAGENS

A primeira função a ser mencionada é a de fila de mensagens. É um modulo intermediário que pode operar no armazenamento temporário de informação ou realizar a tradução do protocolo do remetente para o protocolo do destinatário. É de suma importância para alinhamento de mensagens para que um serviço que funcione de maneira

assíncrona. Mesmo que o serviço consumidor das mensagens fique fora do ar ou apresente uma taxa de consumo dessincronizada com a taxa de envio das mensagens, o uso da fila garante que as mensagens serão consumidas pelo destinatário. Para desempenho de tal função, podemos destacar:

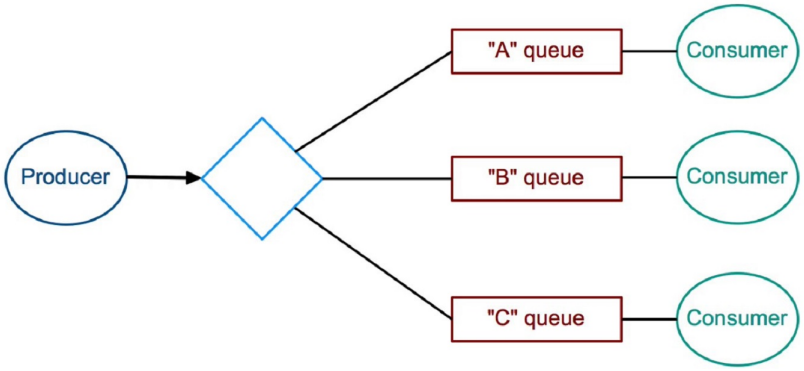
- RabbitMQ: é um projeto *open-source* para intermediação de mensagens escrito na linguagem de programação Erlang. Implementa diversos protocolos como o AMQP, STOMP, Transporte de Telemetria de Enfileiramento (*MQTT*), entre outros protocolos. Opera como servidor e é construído para operar em um grupo de servidores e ser resistente à falhas. Bibliotecas cliente para interface estão disponíveis para a grande maioria das linguagens de programação. Para utilização do RabbitMQ, o consumidor ao se conectar ao serviço de filas instancia uma nova fila que possui um tópico associado à mesma. Para cada consumidor é criada portanto uma fila de envio de mensagens. Quando o remetente enviar uma mensagem que contenha tal tópico ao serviço do RabbitMQ, é direcionado automaticamente estas mensagens às filas dos consumidores. Desta maneira é possível não só a garantia de entrega de um remente a um consumidor, como uma estrutura complexa que permite envio de um ou mais remetentes a um ou mais consumidores, conforme ilutra a figura 4. O serviço também permite o monitoramento e administração das filas criadas (ROSTANSKI; GROCHLA; SEMAN, 2014).

- Redis: é um projeto *open-source* de estrutura de dados na memória, podendo ser classificado também como servidor de dicionário remoto. Implementa um banco de dados de valor-chave distribuído na memória do servidor, sendo possível definir tempo de durabilidade dos dados. O Redis suporta diferentes tipos de estruturas de dados abstratos, como texto, listas, mapas, conjuntos, conjuntos de classificação, índices espaciais. Para suprir a função de fila de mensagens, ele estaria disponível como armazenamento de dados entre o sistema remetente e o sistema de consumo. Neste caso, pode ser desempenhado um papel de banco temporário, uma vez que ao armazenar um dado, ou valor-chave, é possível atrelar um tempo de vida útil do mesmo. Outra vantagem é a rapidez de acesso aos dados pelo fato do mesmo armazenar em memória e não em disco. Porém, não é possível definir inteligência no sistema de maneira a realizar a função de tradução de mensagem (CARLSON, 2013).

2.3.2 BANCO DE DADOS

Para que o sistema opere de forma viável é necessário o armazenamento definitivo dos dados. Para as informações recebidas pelos sistemas externos, que servem como gatilho para os cálculos do sistema, não há necessidade de armazenamento definitivo, apenas a

Figura 4: Fluxo de informação do RabbitMQ



Fonte: (SOFTWARE, 2018)

garantia que o serviço irá receber estas mensagens sem perda ou atrasos. Os bancos de dados constituem parte fundamental da solução pois os resultados dos cálculos precisam ser mantidos para visualização dos usuários e para persistência dos parâmetros de configuração. Há porém vários tipos de banco de dados que podem ser utilizados dos quais podemos citar:

- MongoDB: é um branco de dados orientado ao armazenamento de documentos com plataforma *open-source*. Classificado como banco de dados não relacional (*NoSQL*), ou seja, não apresenta uma estrutura convencional de tabelas para armazenamento dos dados. Os valores são armazenados em uma única coluna de dados, que são dispostos dentro de coleções e organizados pela data de modificação. Dentro de uma mesma coleção é possível existir múltiplas colunas de dados sem a necessidade de atribuir a esta coluna uma variável em específico. Também é possível organizar estas coleções de documento de maneira a definir um tamanho máximo de armazenamento ou número máximo de documentos armazenados. A estrutura não relacional apresenta a grande vantagem de escalamento horizontal, permitindo a adesão de mais servidores operando de maneira conjunta ao invés do aumento da potência de um servidor em específico. Para soluções de pequeno porte esta característica não apresenta tanta importância devido ao baixo fluxo de dados envolvido. O MongoDB também possui gestão de usuários e possibilidade de acesso via servidores externos, como hospedagem na nuvem ou utilização de um outro servidor na mesma rede. Uma comparação rápida pode ser observada na tabela 2 (BANKER, 2011).

- Postgres: é um sistema *open-source* de gerenciamento de banco de dados relacional de objetos com ênfase em extensibilidade e conformidade a padrões. É compatível com

o princípio ACID (*Atomic, Consistent, Isolated, and Durable* - Atômico, consistente, isolado e durável) e transacional, ou seja, possui foco em operações isoladas com um volume contido de dados. Também dá suporte a diversas ferramentas de bancos relacionais como visualizações materializadas, gatilhos de eventos, uso de funções e chave estrangeiras. Suporta uso de vários usuários simultaneamente (OBE; HSU, 2017).

- Influx: é um banco *open-source* de séries temporais desenvolvido em linguagem Go. Provê um serviço otimizado para rápido armazenamento, alta disponibilidade e recuperação de dados das séries temporais. Todo conjunto de dados possui alta compactação mas ainda assim é possível atribuir uma data limite de permanência no banco para economia de espaço em disco. Apesar da compactação uma das premissas do serviço é o acesso rápido a um grande volume de dados temporais através de consultas do tipo *query*, similar a bancos relacionais, com disponibilidade para utilização de expressões regulares (*REGEX*), fórmulas aritméticas ou funções específicas para séries temporais (ANDERSON, 2018).

É complicado decidir se a escolha de um único banco seria a melhor alternativa pois nenhum banco satisfaz completamente as necessidades de uma solução que trabalha com uma grande variedade e volume de dados. O uso de banco de dados relacional é excelente para armazenamento de dados do tipo registro, nos quais é bem simples de imaginar a estrutura de dados que seja estática e armazenável na forma uma tabela. Já para dados não relacionais é aplicável a dados com estrutura dinâmica ou em que seja interessante controle de versão dos documentos armazenados, documentos estes que podem ser arquivos estruturados como documentos de texto, XML ou JSON. Os dados temporais encaixam perfeitamente em dados de ambiente industrial que apresentam a variação ao longo do tempo e permitem a comparação de conjuntos de variáveis. Há ainda a possibilidade de uso do Reddis para operar como armazenamento de dados temporários quando há necessidade de acesso em alta velocidade, por se tratar de um armazenamento em memória. Portanto, a possibilidade de adoção de mais de um banco de dados é altamente viável, uma vez que não há nenhum impedimento para uso de dois bancos simultaneamente(ABRAMOVA; BERNARDINO; FURTADO, 2014).

Relacional	**Não relacional**
Tabela	Coleção
Registro	Documento
Coluna	Atributo
Escalamento vertical	Escalamento horizontal
Estrutura estática	Estrutura dinâmica
Estrutura estática	Estrutura dinâmica

Tabela 2: Comparação entre dados relacionais e não relacionais

2.3.3 LINGUAGEM DE PROGRAMAÇÃO

Ao se construir uma arquitetura utilizando da filosofia de multisserviços, a escolha de uma linguagem de programação padrão não é uma necessidade e não há nenhuma recomendação a respeito de tal padronização. A tendência é que em uma equipe de desenvolvimento de software estejam envolvidas pessoas que possam possuir conhecimento de linguagens diferentes. A restrição pode na verdade limitar a inclusão de novos membros e, pelo fato de novas tecnologias surgirem com frequência, nem todas as linguagens podem estar aptas e limita o horizonte da solução como um todo. Abaixo serão citadas algumas plataformas e suas linguagens que podem ser utilizadas na construção de serviços dentro da solução.

- .NetCore: estrutura de software *open-source* multiplataforma, incluindo o sistema operacional Linux. A linguagem de programação é o CSharp. Funciona com uma biblioteca de aplicações que possui similaridades ao .NetFramework, principal estrutura de software da Microsoft, mas contém seu próprio repositório agindo de forma independente. Oferece suporte a quatro cenários, todos com suporte à multiplataforma: construção de aplicativos web, construção de aplicativos de linha de comando (*Console aplication*), construção de bibliotecas e aplicativos compatíveis a plataforma de sistemas operacionais Windows. Apresenta estrutura modular, ou seja, é possível a inclusão de módulos que são responsáveis por tarefas específicas, inclusão de bibliotecas externas ou comunicação a serviços como banco de dados. Dentro da arquitetura atual seria altamente recomendável para desenvolvimento de serviços que operem no servidor respondendo a requisições HTTP ou que possuam algum gatilho por recebimento de mensagem (BALENA, 2002).
- NodeJs: ambiente de execução para linguagem JavaScript de *open-source* que executa o código fora de um navegador. Historicamente, o JavaScript era usado principalmente para roteiros de código do lado do cliente, nos quais códigos escritos em JavaScript são incorporados no HTML de uma página e executados no lado do cliente diretamente no navegador do usuário. O Node.js permite que os desenvolvedores usem JavaScript para escrever ferramentas de linha de comando e executar roteiros de código no lado do servidor para produzir conteúdo de página da web dinâmico antes que a mesma seja enviada ao navegador . Consequentemente, o Node.js representa um paradigma de unificar o desenvolvimento de aplicativos em torno de uma única linguagem de programação, em vez de linguagens diferentes para serviços do servidor e do cliente. O Node.js tem uma arquitetura orientada a eventos capaz de realizar operações de entrada e saída de forma assíncrona. Essas opções de projeto visam otimizar o rendimento e a escalabilidade em aplicativos da Web com alto tráfego de operações, bem como para aplicativos que realizem comunicação em tempo real (CANTELON et al., 2014).

- Java: linguagem de programação de computadores de uso geral que é baseada em classe e orientada a objetos. É especificamente projetada para ter o menor número possível de dependências de implementação permitindo que os desenvolvedores de serviços gravem uma vez, executem em qualquer lugar. O código Java compilado pode ser executado em todas as plataformas que suportam Java sem a necessidade de recompilação. Isso porque não é utilizado um compilador do sistema, mas ocorre a instalação de uma camada acima da camada do sistema operacional. Esta camada é denominada JVM (*Java Virtual Machine* e torna os aplicativos desenvolvidos em Java independentes da arquitetura do computador. A partir de 2016, Java é uma das linguagens de programação mais populares em uso, especialmente para aplicações web cliente-servidor. Foi originalmente desenvolvido com sintaxe similar às linguagem C e C ++, mas possui menor foco a aplicações de baixo nível, não sendo recomendada a sistemas embarcados (ARNOLD; GOSLING; HOLMES, 2005).

- Phyton: linguagem de programação interpretada de alto nível e uso gratuito. Possui uma filosofia de design que enfatiza a legibilidade do código, notavelmente conhecida pela edentação obrigatória. Fornece construções que permitem uma programação clara em pequenas e grandes escalas. Possui sistema de tipo dinâmico e gerenciamento automático de memória. Suporta múltiplos paradigmas de programação, incluindo orientação a objetos, imperativos, funcionais e processuais. Possui biblioteca padrão ampla e abrangente com interpretação disponível para muitos sistemas operacionais. Há uma linguagem variante ao Phyton, o *CPyhton*, que é um software de *open-source* que tem um modelo de desenvolvimento baseado na comunidade de desenvolvedores (PEDREGOSA et al., 2011).

- R: linguagem de programação e ambiente de software livre para computação estatística e geração de gráficos. Amplamente usada entre estatísticos e mineradores de dados para o desenvolvimento de software voltado à análise de dados. Pesquisas relacionadas à mineração de dados mostram que a popularidade da linguagem R aumentou substancialmente nos últimos anos (TIPPMANN, 2015). O R é um pacote do sistema operacional GNU (*Unix*) com código disponível gratuitamente sob a Licença Pública Geral GNU, e versões binárias pré-compiladas são fornecidas para vários sistemas operacionais. Embora R tenha uma interface de linha de comando, existem serviços com interfaceamentos gráficos e ambientes de desenvolvimento integrados (MATLOFF, 2011).

- C: linguagem de programação de computador imperativa, de propósito geral, com suporte à programação estruturada, recursão e possui sistema de tipo estático que evita muitas operações não intencionais de serem executadas. Fornece construções que mapeiam eficientemente instruções típicas de máquina encontrando uso duradouro em aplicativos que anteriormente tinham sido codificados em linguagem

> muito mais difíceis de interfacear, incluindo sistemas operacionais ou até mesmo sistemas embarcados. C foi originalmente desenvolvido para reimplementação do sistema operacional Unix e tornou-se uma das linguagens de programação mais amplamente utilizadas pois foi projetado para ser compilado usando um compilador relativamente simples, tendo acesso de baixo nível à memória, fornecendo construções de linguagem que mapeiem eficientemente para instruções de máquina e exigir suporte mínimo em tempo de execução. Em um programa C compatível com os padrões pode ser compilado para uma ampla variedade de plataformas de computador e sistemas operacionais com poucas alterações em seu código-fonte. Mesmo não sendo nativamente multiplataforma, a linguagem tornou-se disponível em uma ampla variedade de plataformas, desde microcontroladores embutidos até supercomputadores (KERNIGHAN; RITCHIE, 2006).

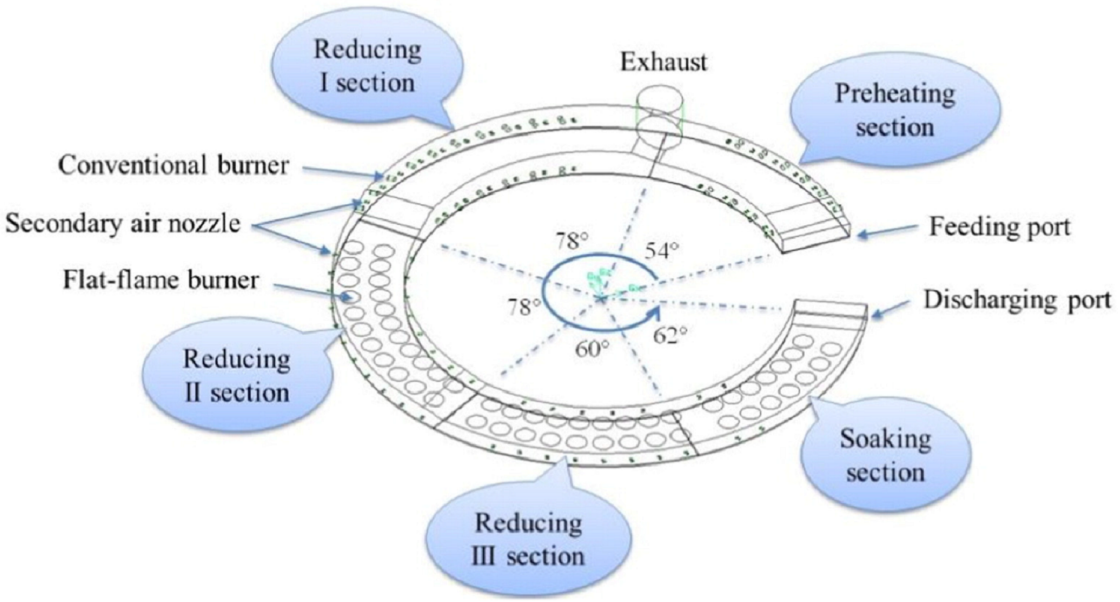
Cada linguagem tem um potencial aplicável às necessidades que um serviço precisa cumprir. Como a arquitetura de micro-serviços não apresenta uma dependência de escolha de um único tipo de linguagem, é possível realizar a escolha ligando às vantagens que cada uma apresenta. Por exemplo, para modelagem matemática pode ser mais viável a utilização da linguagem R. Já para aplicações que exigem maior número de recursos do servidor a escolha da plataforma .NetCore ou Java. Em aplicações que rodem do lado do cliente há maiores vantagens realizar o uso de Javascript com a plataforma NodeJS.

3 DESENVOLVIMENTO

3.1 Estudo de caso - Forno Rotativo

O forno com soleira rotativa é um dos agregados básicos no fluxo tecnológico para a fabricação de tubos sem costura. Sua finalidade é realizar o aquecimento dos blocos desde a temperatura ambiente até a temperatura desejada de laminação. É importante dispor os blocos dentro do forno de maneira uniforme, tanto na distância entre os blocos quanto a distância das extremidades dos blocos em relação às paredes. Desta maneira é garantido que o calor se distribua de maneira uniforme para aquecimento dos blocos de maneira homogênea (MUREŞAN; ABRUDEAN, 2012). O forno possui setores reguladores de temperatura, com queimadores de gás natural dispostos ao longo do teto destes setores. Os setores são categorizados em setor de cargam setores de pré-aquecimento e setor de descarga. Para o fornecimento e evacuação dos blocos, o forno possui máquinas conhecidas como pinças de carregamento e de descarregamento que inserem os materiais longitudinalmente às posições de entrada e saída do forno. Uma representação simples pode ser observada na figura 6 (MUREŞAN; ABRUDEAN, 2012).

Figura 5: Representação das seções de um forno de soleira rotativa



Fonte: (ODA; IBARAKI; ABE, 2006)

As dimensões de um forno de soleira rotativa variam de acordo com o tamanho dos

blocos a serem aquecidos. Para realizar o aquecimento de blocos de aproximadamente 3 metros de comprimento e 30 cm de diâmetro, o raio externo do forno pode alcançar mais de 25 metros. Seguindo estas dimensões, a soleira que realiza o deslocamento dos blocos ao longo do forno teria, portanto, aproximadamente torno de 9 quilômetros de extensão.

Figura 6: Forno de soleira rotativa

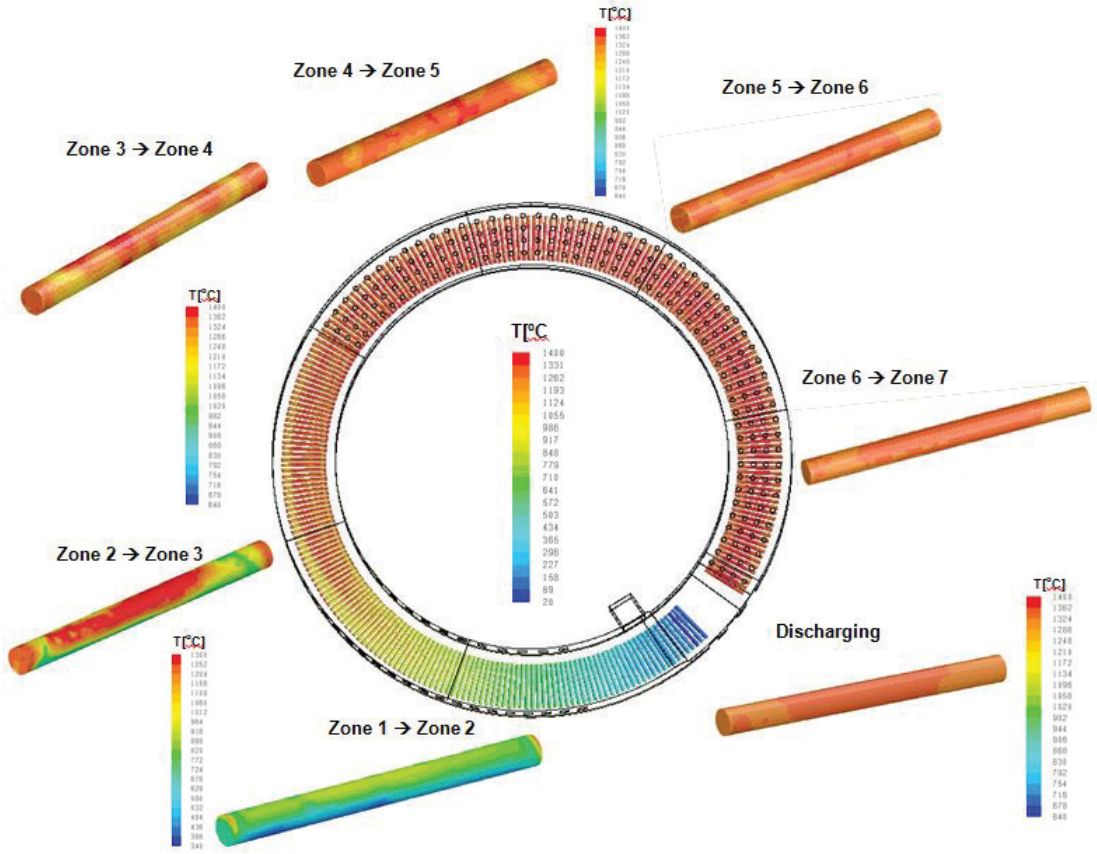
Fonte: (HOTWORK, 2018)

O fornecimento de gás é feito através de tubos do sistema de gás natural que se ramifica ao longo das seções. As temperaturas correspondentes a cada setor são controladas ajustando o fluxo de gás nos tubos correspondentes e com medição realizada através de termopares localizados no teto e lateral dos setores Mureşan e Abrudean (2012). O controle de temperatura consiste portanto em definir uma temperatura que servirá de referência (*SetPoint*) à área exterior dos blocos. Esta definição de temperatura é divida por setor.

O processo de aquecimento dos blocos está diretamente relacionado à temperatura a qual as zonas do forno são aquecidas, ou seja, a área externa dos blocos e também ao tempo de exposição à esta temperatura. O material exposto a este ambiente de alta temperatura vai aos poucos absorvendo energia e se encharcando com o calor até que o bloco apresente uma temperatura uniforme ao longo de todo material. Existem outras variáveis que influenciam diretamente porém com menor intensidade como a temperatura da soleira, temperatura do teto, disposição física dos blocos vizinhos, entre outros motivos. Como a temperatura objetivada de desenfornamento é muito alta, ultrapassando os 1200°C, tais variáveis de menor importância podem ser consideradas irrelevantes por não gerar

uma grande mudança na temperatura final.

Figura 7: Mapa de aquecimento dos blocos ao longo do forno



Fonte: (ROCCA et al.,)

O fator complicador para controle de temperatura de desenfornamento dos blocos dentro do forno é principalmente a dificuldade de mensurar qual a temperatura do bloco ao longo do forno. Uma alternativa é a instalação de termopares ao longo do bloco que realizariam a medição contínua de temperatura. Apesar de ser possível realizar tal instalação, essa alternativa se torna inviável devido a alguns fatores, tais como:

- Altas temperaturas: como as temperaturas objetivadas ultrapassam os 1200 °C, a qualidade dos sensores instalados requer uma blindagem térmica para garantir que a medição será realizada de maneira correta e para prevenir a danificação do equipamento. Alta qualidade é sinônimo de alto custo.

- Quantidade de sensores: é necessário que o bloco seja aquecido de maneira homogênea para que o processo de laminação ocorra sem impacto na qualidade do produto final. Para garantir que os blocos estão sendo mensurados de forma a garantir tal

homogeneidade, é necessário que sejam instalados vários sensores ao longo de cada bloco. Como há centenas de blocos enfornados simultaneamente, a quantidade de sensores a serem instaladas inviabiliza o procedimento de medição da temperatura

- Comunicação: para que os dados mensurados apresentem algum ganho ao controle do forno, é necessário que a informação chegue o mais rápido possível aos operadores responsáveis. Para tal seria necessário obter dados em tempo real dos sensores instalados nos blocos, o que aumentaria o nível de tecnologia investida, sendo outro fator inviabilizador do uso de sensores.

Como não é viável a utilização de sensores, conforme demonstrado nos tópicos acima, uma estratégia é instalar, em um bloco específico, diversos sensores embarcados. Estes sensores armazenam a informação da medição de temperatura de cada ponto do bloco e ao desenfornar, são extraídos os dados para um sistema computacional. Ao realizar a análise destes dados, os mesmos geram uma curva de aquecimento padrão do bloco. Os parâmetros de configuração do forno, tais como velocidade de movimentação da soleira e temperatura aplicada por setor, são gravados e replicados para os demais blocos. Assim, os novos blocos que serão enfornados tem a tendência de apresentar uma curva de aquecimento similar à curva do bloco mensurado anteriormente.

Esta estratégia seria perfeitamente viável a não ser o fato de que pode ocorrer mudanças no ritmo de produção, alterando o tempo de exposição dos blocos à temperatura interna do forno e mudando completamente a curva de aquecimento dos mesmos. Outro fator complicador é que teria a necessidade de que, para todos os blocos, todos parâmetros que foram aplicados ao bloco mensurado precisam ficar estáticos ao longo do percurso de todos os demais blocos. Parâmetros estáticos em um forno com alto consumo de gás e que apresenta regulares alterações no ritmo de produção, implica em uma utilização do forno sem nenhuma otimização dos recursos e, consequentemente, em elevados custos de operação.

A solução para o problema é a construção de um modelo matemático que é responsável por calcular os estados das variáveis do forno e dos blocos enfornados. Uma vez que parâmetros como temperatura do forno por setor, posição atual dos blocos no forno, dados físicos dos blocos e velocidade de movimentação da soleira são conhecidos, é possível estimar a temperatura atual dos blocos. Isso elimina um grande problema que é a dependência da curva padrão de aquecimento. Não é mais necessário seguir um caminho restrito e linear para saber a temperatura atual do bloco, mesmo com tempos de exposição diferentes, mudanças na temperatura dos setores, entre outras imprevisibilidades. Um segundo passo é utilizar da temperatura calculada dos blocos para poder controlar melhor os outros parâmetros do forno. Ao ser desenfornado espera-se que o bloco esteja em uma determinada faixa de temperatura. Como a movimentação da soleira é diretamente

dependente do fluxo dos próximos processos de produção, resta ao modelo controlar a temperatura dos setores. Se o bloco já atingiu a temperatura de desenfornamento não é mais necessário fornecer calor aos blocos e as temperaturas dos setores podem ser reduzidas. Se o bloco porém está com uma temperatura muito baixa também é possível aumentar a temperatura do forno para garantir que ele estará na temperatura desejada ao desenfornar.

O uso portanto de um modelo matemático que possa fornecer tais parâmetros aos operadores responsáveis pelo aquecimento dos blocos, apresentaria um ganho enorme ao processo, permitindo não só a garantia de que o bloco será desenfornado em uma temperatura mais próxima do desejado, independente das inconsistências durante o processo de aquecimento, como também garante que não haverá desperdício de recursos ao sobreaquecer os blocos além da temperatura desejada.

Como o foco deste trabalho é a apresentação de um *MVP* e sua arquitetura que cumpra com a tarefa de inserção de ferramentas e softwares de maneira construtiva, não será abordado em detalhes como seria o comportamento deste modelo matemático para estimar a temperatura de blocos em um forno rotativo. Uma maior abordagem pode ser encontrada em (MUREŞAN; ABRUDEAN, 2012).

O desafio é realizar a inclusão de um modelo matemático para predição da temperatura dos blocos no forno sem que haja impacto negativo no controle do forno rotativo já existente. Ao utilizar o modelo será possível fornecer dados que auxiliaram diretamente na decisão de controle do forno, gerando economia ao otimizar o uso de recursos, neste caso o gás natural, além de garantir que os blocos desenfornados estejam de acordo com os critérios estipulados para o processo de laminação.

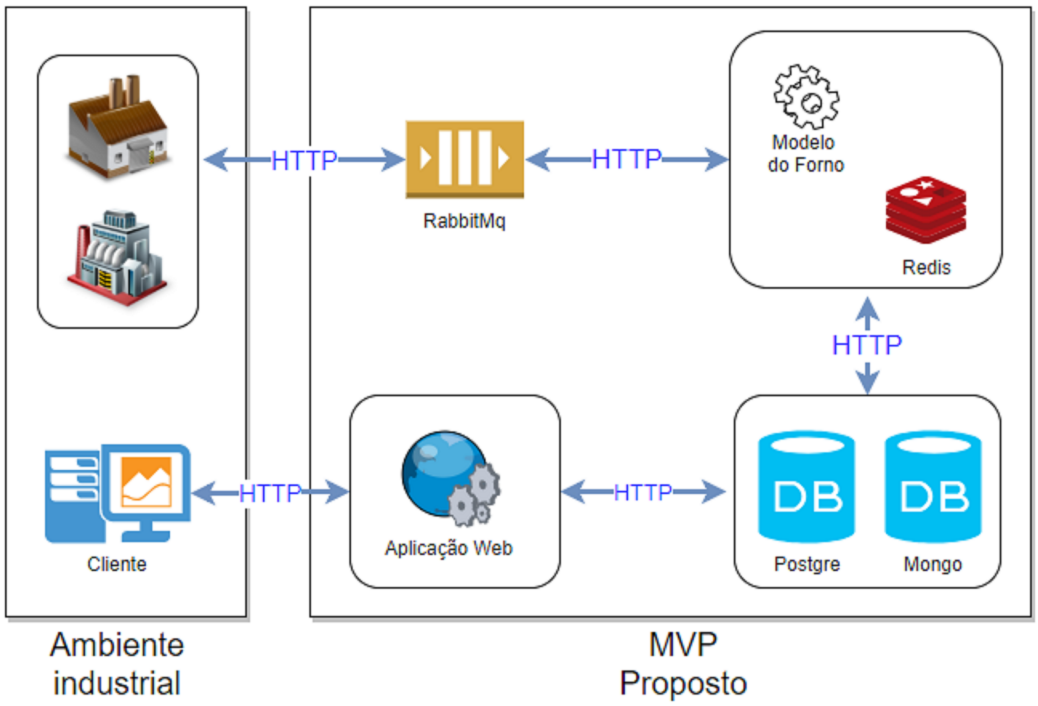
3.2 Arquitetura proposta

Para explicação de como foi feita a arquitetura proposta, é necessário dividir a mesma em algumas seções. Na figura 8 consta uma visão geral do funcionamento da arquitetura. Para melhor entendimento de como funciona o fluxo de dados, desde de os sistemas de origem até o cliente final, esta seção será dividida nos seguintes tópicos: sistema operacional, fila de mensagens, aplicação do modelo matemático, banco de dados e aplicação Web.

3.2.1 Sistema operacional

O sistema operacional adotado é o Linux, por se tratar de um sistema operacional de uso gratuito, o que está alinhado com a demanda de uso de softwares *open-source*. Além disso, o Linux apresenta tanto grande estabilidade para sistemas com alto nível de disponibilidade e escalabilidade, quanto otimização e compactação. Isso é justificado na escolha do Linux tanto em servidores para funcionamento de serviços na nuvem, que é

Figura 8: Visão geral da arquitetura proposta



altamente escalável, até ao ponto de ser utilizado em sistemas com limitação de recursos, como por exemplo sistemas embarcados.

A versão utilizada é o Ubuntu 18.04 LTS. Há outras versões do Linux disponíveis para uso neste tipo de aplicação, tais como Debian, RedHat, entre outros. Esta versão foi escolhida por apresentar grande estabilidade nas versões de suporte prolongado, por estar apta ao uso de todos os softwares listados na seção de softwares propostos e também por possuir apoio continuo da comunidade de desenvolvedores (TORKAR; MINOVES; GARRIGÓS, 2011).

Este sistema operacional pode ser instalado em uma máquina que esteja dentro da rede industrial ou em uma rede externa. A opção de instalação em uma rede interna faz com que a comunicação ocorra de maneira mais segura pois não haveria um tráfego de dados em ambientes externos à empresa. Para tal não seria necessário aumentar o nível de segurança como um todo da rede da empresa apenas para atender à implementação desta aplicação. Se o sistema for instalado em uma máquina externa, apesar da desvantagem da segurança, poderá ser utilizada diversas opções. Tais opções incluem a utilização de hospedar o servidor na nuvem, o que não prejudica o funcionamento do sistema uma vez que toda comunicação pode ser realizada obedecendo à arquitetura REST, mesma arquitetura aplicada em diversos serviços Web. A decisão final parte da necessidade e dos recursos disponíveis para se hospedar o produto, pois, do ponto de vista da arquitetura, as diferenças não são critérios de impedimento da implementação.

3.2.2 Fila de mensagens

O sistema precisa ser capaz de receber diversas informações dos sistemas de produção em tempo real. Essa informação pode ser recebida ao abrir uma porta de comunicação com os outros sistemas de produção. Este tráfego de informação dependeria da construção, dentro do sistema de produção atual, de um módulo capaz de enviar e receber informações. Para satisfazer as teorias de micro serviços, o módulo não poderia interferir em outros processos vigentes do ambiente de produção. Este é o único ponto negativo da proposta em relação aos objetivos, uma vez que há dependência de instalação no sistema de produção vigente para que haja um correto funcionamento do modelo matemático.

Uma vez instalado o módulo, as informações passarão a ser recebidas e enfileiradas pelo serviço do RabbitMQ, citado na seção de revisão de literatura. O RabbitMQ garante que toda informação vinda do sistema de produção será armazenada temporariamente até que o serviço ligado ao modelo matemático realize o consumo destes dados. Este serviço, lê os dados do RabbitMQ e aplica um filtro de informações da produção para que o modelo seja alimentado com as informações necessárias.

Para que o cálculo seja feito pelo serviço relacionado ao modelo matemático, é necessário garantir que as informações cheguem a este serviço de maneira síncrona. O sincronismo é muito importante pois não se pode permitir que os dados estejam desalinhados em relação ao tempo. Desta maneira, os dados representam de forma leal os estados dos blocos ao longo do forno no processo de aquecimento.

3.2.3 Aplicação do modelo matemático

Analisando a complexidade de um modelo para um forno rotativo, é viável o armazenamento temporário de informações. Assim o modelo não precisa armazenar valores de cálculo internamente e possibilita até mesmo a realização de novas chamadas do modelo caso haja alguma exceção durante o cálculo. Este armazenamento pode ser feito utilizando o software *Reddis* citado na seção de revisão. Com este armazenamento é possível obter melhor performance e gera maior robustez a erros de cálculo do modelo.

Após o cálculo do modelo ser realizado com sucesso, o resultado precisa ser armazenado em um banco de dados para persistência de informação. Uma estratégia mais comum é permitir que o mesmo programa que realize o cálculo já faça acesso diretamente ao banco de dados para salvar os arquivos. Mas essa abordagem não é interessante pois agrega dois papéis a um mesmo serviço. Para evitar que isso ocorra, é criado um novo serviço cuja função é unicamente estabelecer comunicação com os serviços de banco de dados. A estratégia de criar um serviço isolado é para não ferir o princípio de organização descentralizada de micro serviços. Assim este módulo consegue receber e passar informações a outros módulos, independente da estrutura do banco ou de qual banco está sendo

utilizado. Para os outros serviços não é interessante ser dependente de um banco específico, de onde este banco está instalado e dos pormenores para que o acesso seja feito com sucesso. Como já foi discutido no capítulo 2, há vantagens e desvantagens entre escolher um banco relacional ou não-relacional para salvar os dados. Compete a este novo serviço realizar o armazenamento dos dados e eliminar essa camada de inteligência dos demais módulos. Como padrão *REST* a comunicação deste serviço com os demais é realizada através de requisições HTTP.

Portanto, é incumbido a este serviço responder às requisições de salvamento dos resultados enviados pelo serviço que realiza integração do modelo matemático e enviar para outros serviços requisições de acesso aos dados salvos. O serviço responsável por exibir os dados aos usuários que pode requisitar tais resultados.

3.2.4 Cliente para exibição

Uma vez que os dados de cálculo estão armazenados, é possível exibir para os usuários do sistema industrial de maneira gráfica e amigável todas as informações referentes ao forno rotativo, como temperatura dos blocos presentes no forno ou qual a temperatura indicada para cada seção. A maneira como isso é feita é desenvolvendo um novo serviço responsável por requisitar as informações do módulo de banco de dados e que fornece os dados à um cliente conectado por plataforma web, por meio de um navegador.

Através deste acesso é possível que o operador observe o comportamento de aquecimento dos blocos no forno em tempo real. Assim, pode-se alterar as configurações de temperatura das zones do forno conforme novas situações forem surgindo, de maneira que seja garantido o aquecimento dos blocos com uso otimizado de recursos.

Como os dados estão persistentes no banco de dados, é possível também o desenvolvimento de telas que possibilitem a visualização dos dados de produção de blocos que já foram desenfornados. Desta maneira, outros setores como o setor de qualidade da produção podem usufruir de tais dados para realizar análises do processo de produção e melhorar a qualidade do material final.

4 DISCUSSÃO

Esta proposta de produto minimamente viável foi apresentada afim de saciar o objetivo geral, que é o desenvolvimento de ferramentas alinhadas com às diretrizes da industria 4.0 e que apresentem ganho sem interferência destrutiva no processo.

Tal objetivo foi alcançado ao se desenvolver serviços que são capazes de realizar comunicação com outros sistemas que já estão em operação, com mínima interferência, ao desenvolver todos os serviços fazendo utilização de sistemas e softwares *open-source*, aplicando teorias de micro-serviços para a estruturação de uma arquitetura que disponibiliza ao usuário final um interface com dados muito relevantes ao processo de produção, de forma a otimizar uso de recursos sem impactar na qualidade do material produzido.

Ao se definir uma arquitetura que permita a construção de ferramentas e serviços acopláveis à solução geral, e estes serviços adicionados não apresentam relação de dependência em relação aos demais serviços presentes, se torna possível a inclusão de diversas ferramentas independentes sem que haja impacto na solução geral como um todo. Outras soluções padrões na área de automação, como sistemas supervisórios, não seriam capazes de contemplar os objetivos alcançados sem que haja mudança profunda na estrutura do projeto ou a adaptação não proveria escalabilidade para melhorias futuras.

Em relação ao estudo de caso do forno de soleira rotativa, a utilização do produto desenvolvido proveu aos operadores mais controle e análise crítica do processo. Houve maior segurança ao realizar o desenfornamento para o processo de laminação, pois os requisitos de temperatura definidos agora são claramente monitorados devido ao acompanhamento em tempo real da temperatura dos blocos. Se algum bloco não atende aos requisitos de temperatura para desenfornamento, já é possível ao operador do forno intervir no processo e esperar que haja aquecimento para que tal condição seja atendida antes de desenfornar o bloco. Em situação contrária, caso um bloco já tenha atingido os critérios de desenfornamento antes da posição de desenfornamento, é possível realizar a redução de temperatura das zonas de aquecimento e reduzir o consumo de gás no forno, consequentemente gerando economia na operação do forno.

Do ponto de vista da qualidade do material, a possibilidade de consultar o histórico de aquecimento dos blocos se torna uma excelente ferramenta. Caso um material específico não apresente uma qualidade final satisfatória, é possível analisar como ocorreu o aquecimento, qual foi a temperatura do bloco ao desenfornar e concluir se o processo de aquecimento realmente teve um impacto negativo ao se investigar falhas no processo

de laminação. Também se torna possível otimizar o funcionamento geral do forno, reanalisando as configurações de aquecimento das zonas ao longo do tempo, para diferentes tipos de materiais, diferentes condições de operação. Este tipo de análise não deve ser feita pelo operador do forno, pois o mesmo se restringe a fazer a análise em tempo real. Disponibilizar uma grande amostra de dados ao longo do tempo à uma equipe com maior capacidade técnica, propicia uma análise profunda e crítica do processo para que sejam tomadas melhores decisões com o objetivo de otimizar o funcionamento do forno.

Para melhorar a utilização do produto, é necessário realizar esforços em diversos pontos tais como:

- Validação dos dados: Ao se realizar o cálculo da temperatura dos blocos, o modelo matemático é capaz de fornecer inúmeros dados em relação ao processo de aquecimento. É necessário portanto uma resposta dos especialistas de quais valores agregam valor ao processo para análise da produção, geração de gráficos e relatórios;
- *Feedback* dos usuários: disponibilizar aos usuários um canal de comunicação aberto em que os mesmos possam expor pontos de melhoria nas interfaces e sugerir melhorias gerais na exibição dos dados;
- Melhoria no funcionamento do modelo: realizar testes junto com a produção da eficácia de cálculo de temperatura do modelo e qual a faixa de incerteza dadas as configurações possíveis.

Do ponto de vista do controle, uma vez que já é possível prever as temperaturas dos blocos ao longo do forno, pode ser possível incumbir ao modelo o papel de controle de temperatura das zonas do forno. Com o cálculo da temperatura dos blocos e a sugestão de temperatura das zonas de aquecimento do forno, é possível fechar um fluxo em que a cada iteração seria possível observar o impacto desta sugestão do modelo no comportamento térmico do forno e do aquecimento dos blocos. Deixar o modelo mais inteligente proporciona maior conforto ao operador, que obtêm uma fonte de consulta para quais temperaturas deve-se configurar o forno.

Conclusão

Ao adotar de sistemas que sejam compostos por novas tecnologias em um ambiente industrial que seja composto por inúmeros outros sistemas, é fundamental que seja preservado um bom comportamento deste sistema e que não haja interferência destrutiva nos demais processos.

Dentro de uma solução que utilize a arquitetura de micro-serviços é possível realizar o desenvolvimento independente da linguagem de programação visto que cada serviço pode desempenhar funções mais específicas e, ao utilizar um protocolo de comunicação padrão como proposto pelo REST, não apresentar interdependência entre estes sistemas. Em um ambiente de restritas possibilidades essa liberdade pode facilitar o desenvolvimento de serviços bem como a inclusão de novos membros, removendo barreiras de aprendizado e de limitações de tecnologia.

Neste *MVP* proposto, para a arquitetura adotada foi necessária uma pequena interferência para que haja possibilidade de comunicação dos dados entre os sistemas de produção e o modelo referenciado como estudo de caso. É importante analisar estes casos para que haja melhor entendimento das limitações e problemas que a implantação de um sistema como este possa causar em um ambiente industrial. Estar preparado para integração à indústria 4.0 é crucial para que sejam desenvolvidos outros serviços e recursos ao longo prazo.

No estudo de caso do forno de soleira rotativa, o produto apresentado obteve êxito do ponto de vista do controle do forno e supervisão das temperaturas por parte dos operadores. Como melhoria para trabalhos futuros é necessário estabelecer contato com os usuários para que novas sugestões sejam feitas para interfaces e também realizar revisões por parte dos resultados do modelo de forma a melhorar a precisão dos cálculos feitos.

Referências

ABRAMOVA, V.; BERNARDINO, J.; FURTADO, P. Experimental evaluation of nosql databases. *International Journal of Database Management Systems*, 2014. Academy & Industry Research Collaboration Center (AIRCC), v. 6, n. 3, p. 1, 2014. Citado na página 37.

ANDERSON, S. *InfluxDb 1.6 Release Notes*. 2018. Disponível em: <https://docs-.influxdata.com/influxdb/v1.6/about_the_project/releasenotes-changelog/>. Citado na página 37.

ARNOLD, K.; GOSLING, J.; HOLMES, D. *The Java programming language*. [S.l.]: Addison Wesley Professional, 2005. Citado na página 39.

BALENA, F. *Programming microsoft visual basic. NET (Core Reference)*. [S.l.]: Microsoft Press, 2002. Citado na página 38.

BANKER, K. *MongoDB in action*. [S.l.]: Manning Publications Co., 2011. Citado na página 36.

BREWER, E. A. Towards robust distributed systems. In: *PODC*. [S.l.: s.n.], 2000. v. 7. Citado na página 33.

CANTELON, M. et al. *Node. js in Action*. [S.l.]: Manning Greenwich, 2014. Citado na página 38.

CARLSON, J. L. *Redis in action*. [S.l.]: Manning Publications Co., 2013. Citado na página 35.

DMITRY, N.; MANFRED, S.-S. On micro-services architecture. *International Journal of Open Information Technologies*, 2014. v. 2, n. 9, 2014. Citado 3 vezes nas páginas 32, 33 e 34.

DOUKAS, G.; THRAMBOULIDIS, K. A real-time-linux-based framework for model-driven engineering in control and automation. *IEEE Transactions on Industrial Electronics*, 2011. IEEE, v. 58, n. 3, p. 914–924, 2011. Citado na página 34.

FIELDING, R. T.; TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. Citado na página 28.

GROUP, N. W. et al. Rfc 2616 hypertext transfer protocol–http/1.1. *R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee*, 1999. 1999. Citado 2 vezes nas páginas 27 e 29.

HOTWORK. *HOTWORK SOLUTIONS FOR FURNACE DESIGNERS*. 2018. Disponível em: <http://www.hotwork.com/industry-solutions/furnace-designers/>. Citado na página 42.

KERNIGHAN, B. W.; RITCHIE, D. M. *The C programming language*. [S.l.: s.n.], 2006. Citado na página 40.

LASI, H. et al. Industry 4.0. *Business & Information Systems Engineering*, 2014. Springer, v. 6, n. 4, p. 239–242, 2014. Citado na página 23.

LEWIS, J.; FOWLER, M. Microservices: a definition of this new architectural term. *Mars*, 2014. 2014. Citado na página 31.

MATLOFF, N. *The art of R programming: A tour of statistical software design*. [S.l.]: No Starch Press, 2011. Citado na página 39.

MUREŞAN, V.; ABRUDEAN, M. The control of the billets heating process in a furnace with rotary hearth. *IFAC Proceedings Volumes*, 2012. Elsevier, v. 45, n. 21, p. 735–740, 2012. Citado 3 vezes nas páginas 41, 42 e 45.

NEWMAN, S. *Building microservices: designing fine-grained systems*. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado na página 31.

OBE, R. O.; HSU, L. S. *PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database*. [S.l.]: "O'Reilly Media, Inc.", 2017. Citado na página 37.

ODA, H.; IBARAKI, T.; ABE, Y. Dust recycling system by the rotary hearth furnace. *Shinnittetsu Giho*, 2006. NIPPON STEEL CORPORATION, v. 384, p. 134, 2006. Citado na página 41.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 2011. v. 12, n. Oct, p. 2825–2830, 2011. Citado na página 39.

ROCCA, A. D. et al. Rotary hearth furnace advanced design methods. Citado na página 43.

ROSTANSKI, M.; GROCHLA, K.; SEMAN, A. Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq. In: IEEE. *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. [S.l.], 2014. p. 879–884. Citado na página 35.

SHAH, R.; WARD, P. T. Lean manufacturing: context, practice bundles, and performance. *Journal of operations management*, 2003. Elsevier, v. 21, n. 2, p. 129–149, 2003. Citado na página 30.

SOFTWARE, I. P. *Data flow diagram of RabbitMQ*. 2018. Disponível em: <https://www-.rabbitmq.com/documentation.html>. Citado na página 36.

THÖNES, J. Microservices. *IEEE software*, 2015. IEEE, v. 32, n. 1, p. 116–116, 2015. Citado na página 30.

TIPPMANN, S. Programming tools: Adventures with r. *Nature News*, 2015. v. 517, n. 7532, p. 109, 2015. Citado na página 39.

TORKAR, R.; MINOVES, P.; GARRIGÓS, J. Adopting free/libre/open source software practices, techniques and methods for industrial use. *Journal of the AIS*, 2011. Association for Information Systems, v. 12, n. 1, 2011. Citado na página 46.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. *REST in practice: Hypermedia and systems architecture*. [S.l.]: "O'Reilly Media, Inc.", 2010. Citado na página 27.

Certifico que o aluno Glauco de Oliveira Leme, autor do trabalho de conclusão de curso intitulado "Desenvolvimento de Produto Minimamente Viável Para Sistemas Industriais de Automação", efetuou as correções sugeridas pela banca examinadora e que estou de acordo com a versão final do trabalho.

Adrielle de Carvalho Santana, Orientador