

CENTRO UNIVERSITÁRIO UNIDADE DE ENSINO SUPERIOR DOM BOSCO
CURSO SISTEMAS DE INFORMAÇÃO

ANTONIO JOAQUIM E SILVA FILHO

**ANÁLISE DE MODELO DE APLICAÇÕES DE ALTA DISPONIBILIDADE: UM
ESTUDO DA APLICABILIDADE DAS TECNOLOGIAS DOCKER E KUBERNETES**

São Luís

2020

ANTONIO JOAQUIM E SILVA FILHO

**ANÁLISE DE MODELO DE APLICAÇÕES DE ALTA DISPONIBILIDADE: UM
ESTUDO DA APLICABILIDADE DAS TECNOLOGIAS DOCKER E KUBERNETES**

Monografia apresentada ao Curso de Sistemas de Informação do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Allisson Jorge Silva Almeida.

São Luís

2020

Silva Filho, Antonio Joaquim e

Análise de modelo de aplicações de alta disponibilidade: um estudo da aplicabilidade das tecnologias Docker e Kubernetes/ Antonio Joaquim e Silva Filho. São Luís, 2020.

40f.

Orientador: Prof. Me. Allisson Jorge Silva Almeida.

Monografia (Graduação em Sistema de Informação) - Curso de Sistema da Informação – Centro Universitário Unidade de Ensino Superior Dom Bosco – UNDB, 2020.

1. Desenvolvimento de software. 2. Docker - Kubernetes. 3. Micro serviços. I. Título.

CDU 004.41

ANTONIO JOAQUIM E SILVA FILHO

ANÁLISE DE MODELO DE APLICAÇÕES DE ALTA DISPONIBILIDADE: UM ESTUDO DA APLICABILIDADE DAS TECNOLOGIAS DOCKER E KUBERNETES

Monografia apresentada ao Curso de Sistemas de Informação do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovada em: ____/____/____.

BANCA EXAMINADORA:

Prof. Me. Allisson Jorge Silva Almeida

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Prof. Esp. Pedro Henrique Carneiro Gomes

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Prof. Me. Me. Rafael de Souza Cunha

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Dedico primeiramente a Deus e
a minha família.

AGRADECIMENTOS

Agradeço ao meu orientador professor MSc. Allisson Jorge Silva Almeida e ao professor MSc. Rafael de Souza Cunha, juntamente com todos os membros da banca, o coordenador dos cursos de tecnologia da informação Rodrigo Monteiro de Lima, além dos meus excelentes professores do curso de Sistemas de Informação, por me ajudarem a vencer mais um obstáculo em minha vida.

RESUMO

Este trabalho tem como objetivo auxiliar os profissionais que queiram utilizar e desenvolver softwares com recursos de alta disponibilidade e escalabilidade, por meio da utilização dos recursos de Micro Serviços, *Docker* e *Kubernetes*. A ideia é analisar modelos, e verificar as vantagens e desvantagens dessas novas tecnologias, e de como utilizá-las em projetos, quando se trata de alta disponibilidade, escalabilidade de recursos tecnológicos de infraestrutura ou software. Recomenda-se a aplicação destas novas tecnologias de containerização com o *Docker*, e gerenciamento com o *Kubernetes*, quando se deseja ter economia com os gastos em recursos tecnológicos, financeiros e mão de obra no âmbito computacional na tecnologia da Informação (TI).

Palavras-chave: Docker. Kubernetes. Alta Disponibilidade. Micro Serviços.

RÉSUMÉ

Ce travail vise à aider les professionnels qui veulent utiliser et développer des logiciels avec des ressources de haute disponibilité et d'évolutivité, grâce à l'utilisation de micro-services, docker et kubernetes ressources. L'idée est d'analyser les modèles, et de vérifier les avantages et les inconvénients de ces nouvelles technologies, et comment les utiliser dans les projets, quand il s'agit de haute disponibilité, l'évolutivité des ressources technologiques de l'infrastructure ou des logiciels. Il est recommandé d'appliquer ces nouvelles technologies de conteneurisation avec Docker, et la gestion avec Kubernetes, lorsque vous voulez avoir des économies avec les dépenses sur les ressources technologiques, financières et de main-d'œuvre dans le domaine informatique dans les technologies de l'information (IT).

Mots-clés: Docker. Des Kubernetes. Haute disponibilité. Micro Services.

LISTA DE FIGURAS

- Figura 1 Docker hub e as imagens de contêineres disponíveis
- Figura 2 Comparação entre Docker e Virtualização
- Figura 3 Comparativos dos três ambientes tradicional, virtualização e contêiner
- Figura 4 Componentes Kubernetes
- Figura 5 Comparativos do sistema monolítico e micros serviços
- Figura 6 Tela VM Linux instalando Docker
- Figura 7 Tela VM Linux visualizando versão do Docker
- Figura 8 Tela VM Linux Visualizando contêineres instalados
- Figura 9 Tela VM Linux mostra os containers executando no Docker
- Figura 10 Tela VM Linux Instalando Kubernetes
- Figura 11 Tela VM Linux Instalando Ferramentas Kubernetes
- Figura 12 Tela VM Linux Verificando Versão Kubeadm
- Figura 13 Tela VM Linux Análise das Aplicações

LISTA DE TABELAS

Tabela 1 Comparativo de Horário de Chegada dos E-mails

LISTA DE ABREVIATURAS E SIGLAS

HOST	Máquina ou Computador
ERP	Enterprise Resource Planning (Planejamento de Recursos Corporativos)
DNS	Domain Name System (Sistema de Nomes de Domínio)
IP	Internet Protocol (Protocolo de Internet)
SSH	Secure Shell (Protocolo de Segurança de Rede)
TI	Tecnologia da Informação
DEPLOY	Implantar, Colocar em posição, disponibilizar para uso
HTTP	HiperText Transference Protocol (Protocolo de transferência de Hiper Texto)
VM	Virtual Machine (Máquina Virtual)
OS	Operating System (Sistema Operacional)
VPS	Virtual Private Server (Servidor Virtual Privado)
MAC	Media Access Control (Controle de Acesso de Mídia)
PODS	São as menores unidades implantáveis de computação que você pode criar e gerenciar em Kubernetes
NODE	Um nó é uma máquina de trabalhadores em Kubernetes

SUMÁRIO

1 INTRODUÇÃO	12
1.1 Objetivo Geral.....	13
1.2 Específico	13
Utilização da tecnologia <i>Docker</i>	13
Utilização da tecnologia <i>Kubernetes</i>	13
Apresentar as vantagens da Alta disponibilidade	13
Analisar os modelos de Aplicações	13
1.3 Delimitações do trabalho.....	13
1.4 Justificativa	13
1.5 Problema.....	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 Docker.....	15
2.2 Kubernetes	17
2.3 Contextualização.....	22
2.4 Alta Disponibilidade.....	22
2.5 Micro Serviços	23
3 MODELO DE APLICAÇÃO DE ALTA DISPONIBILIDADE	25
3.1 Instalação do Docker	25
3.2 Instalação do Kubernetes.....	27
3.3 Escolhendo o Modelo de Aplicação	29
3.4 Levantamento do Requisito da Aplicação.....	29
3.5 Implementação da Aplicação	29
4 RESULTADOS E DISCUSSÕES	30
4.1 Configuração do Ambiente da Aplicação.....	30
4.2 Implantação e Manutenção de Aplicação.....	30
4.2 Tempo de Execução da Aplicação.....	30
5 CONCLUSÃO	33
5.1 Trabalhos Futuros.....	33
REFERÊNCIAS	34
ANEXO A – Programa Fontes da Aplicação não <i>Docker</i> App	36
ANEXO B – Programa Fontes da Aplicação <i>Docker</i> Apptcc	37
ANEXO C – Arquivo de definição <i>Dockerfile</i> Apptcc	38
ANEXO D – Arquivo do Projeto Apptcc	39

1 INTRODUÇÃO

A evolução do desenvolvimento de software vem se modernizando, conforme a necessidade e a tecnologia utilizada a cada etapa do tempo em que é aplicada, algumas metodologias surgiram e deram suporte a esses novos jeitos de pensar, e a construção de soluções tecnológica aos negócios das empresas.

Essa evolução chegou à necessidade da alta disponibilidade de recursos computacionais, que requer uma variedade homogênea de recursos para suprir a eventual perda de alguns deles, e mesmo assim os serviços continuem disponíveis, onde uma pequena empresa possa usufruir da continuidade desses serviços computacionais oferecidos pelas tecnologias dos contêineres *Docker* e orquestração de contêiner *Kubernetes*.

A criação de uma modelo de aplicação de software que atenda às necessidades das empresas de pequeno porte, utilizando os recursos de alta disponibilidade nas tecnologias de contêiner, que possa resolver às necessidades é uma nova realidade e um desafio para os desenvolvedores de software, que querem evitar problemas como a incompatibilidade e dependências de componentes como dlls, drives além de indisponibilidade de recursos e serviço computacional, ainda a incompatibilidade do ambiente de desenvolvimento com o de produção, e permitir que aplicação possa ser migrada para outra máquina com o serviço de contêiner *Docker* sem a necessidade de compilação da aplicação, tornando mais rápido e pratico a sua exportação.

Com esse modelo queremos ajudar os desenvolvedores que ainda não se utilizam desta tecnologia, a uma ajuda a seguir para que possa se orientar e tomar uma decisão em como proceder ao partir para utilização deste novo modelo de tecnologia de desenvolvimento de software em contêiner e sua orquestração.

Demonstrando que essa nova tecnologia se aplica a todas as categorias e modalidade de software principalmente os que requerem de alta disponibilidade, sendo para uma pequena empresa ou uma de médio ou grande porte.

1.1 Objetivo Geral

Uma análise comparativa dos ambientes de softwares convencionais, com o modelo de implementação baseado no ambiente *Docker* e *Kubernetes*.

1.2 Específico

Utilização da tecnologia *Docker*

Utilização da tecnologia *Kubernetes*

Apresentar as vantagens da Alta disponibilidade

Analisar os modelos de Aplicações

1.3 Delimitações do trabalho

Este trabalho do modelo de aplicação com alta disponibilidade, delimita-se a utilização de contêiner com o *Docker* de propriedade da empresa Docker Inc., como infraestrutura na comparação do modelo tradicional de ambiente de software com o de containerização.

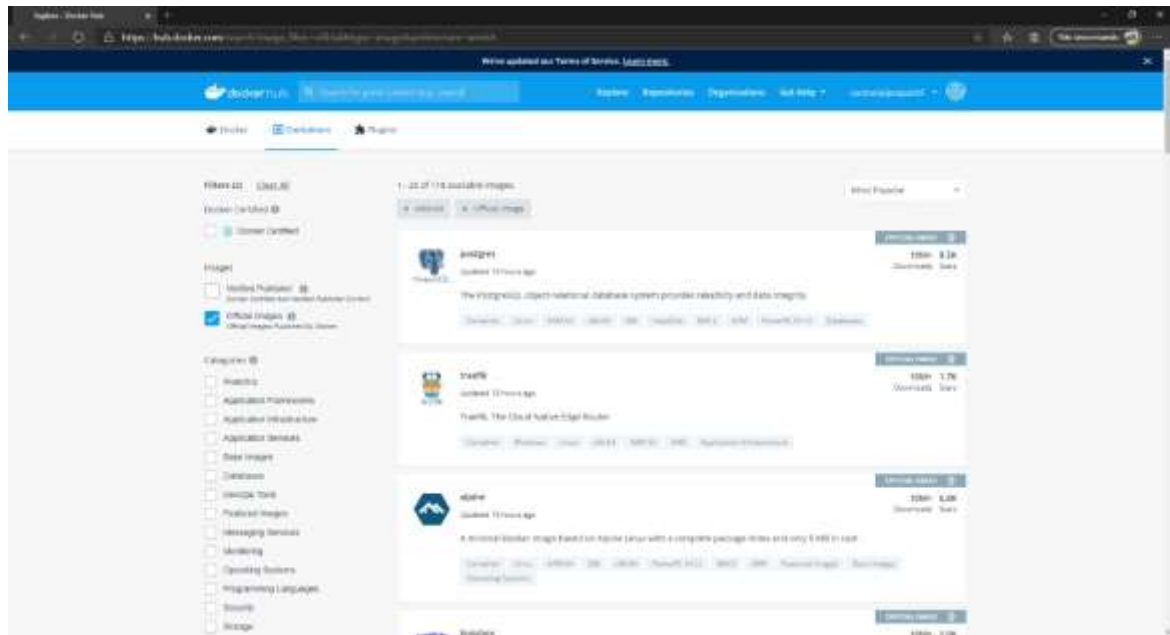
1.4 Justificativa

A utilização do *Docker*, facilita o isolamento de um software ou ambiente completo dele dentro de um container, e a partir desse momento há a portabilidade para qualquer outro host que contenha o Docker instalado se torna possível.

E com isso reduzindo significativamente o tempo de *deploy* da infraestrutura ou até mesmo aplicação, pois não há mais a necessidade de ajustes de ambientação para o funcionamento dos serviços, pós a configuração inicial já foi realizada anteriormente (CRISTIANO, 2015).

Para um melhor aproveitamento dos recursos tecnológicos da empresa e economia em manutenção de software e hardware a utilização da tecnologia Docker é fundamental para otimização de recursos, pós as imagens oficiais públicas de contêineres estão disponíveis no hub Docker como o Apache HTTP Server, MariaDB, mysql, node, e muitos outros disponíveis para utilização conforme figura 1 abaixo.

Figura 1 – Docker hub e as imagens de contêineres disponíveis



Fonte: (DOCKERHUB, 2020).

Não só na economia com software, mas a economia do tempo de implantação e replicação de um contêiner é muito superior a qualquer outra já existente, pós basta uma linha de comando para subir uma imagem de contêiner.

Os problemas mais comuns em uma instalação de um novo software, quando em ambiente de teste, funciona sem problemas, mas já ao entrar em produção, ocorrem diversos tipos de incompatibilidades, como banco de dados ou sistema operacional em idioma diferente, além das bibliotecas e dlls desatualizadas, que geralmente ocasiona perda de tempo ao tentar se identificar o problema, além da mão de obra no retrabalho pra corrigir os erros de incompatibilidade com outros softwares já existente em produção.

1.5 Problema

Analisar se as aplicações e suas dependências como bibliotecas e dlls desenvolvidas com tecnologias tradicionais para pequenas empresas em ambiente de rede local, se aplicam ao novo modelo de << isolamento >>, utilizando a plataforma *Docker* e com gerenciamento desses contêineres com o *Kubernetes*.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão abordados com apoio das literaturas pesquisadas, os principais temas que formam a base teórica deste trabalho: tipo descritivo, explicativo a respeito das tecnologias *Docker* e *Kubernetes*.

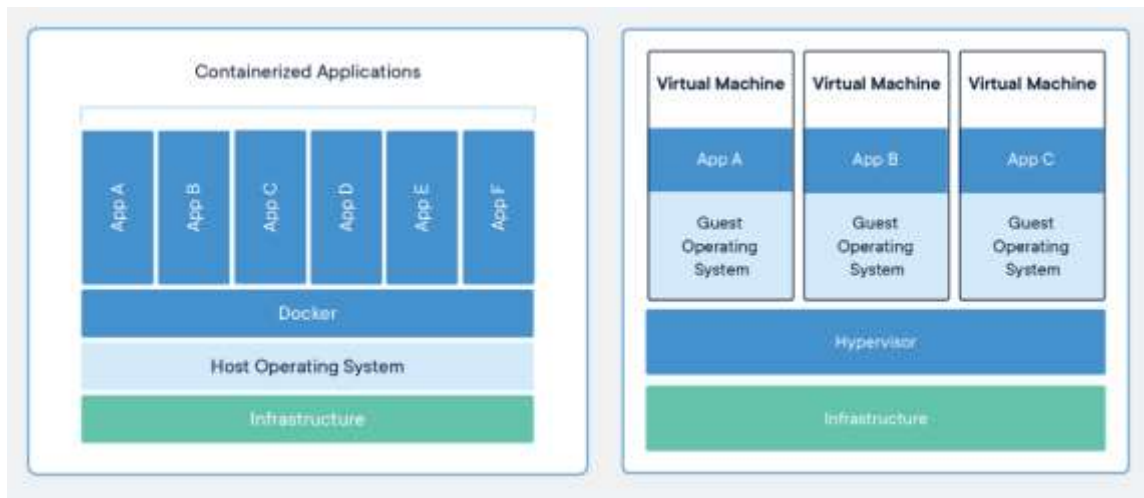
2.1 Docker

A tecnologia de contêiner *Docker* foi lançada em 2013 como um *Docker Engine* de código aberto, *Docker* é uma plataforma de (*Open Source*), que permite o empacotamento de uma aplicação e suas dependências como bibliotecas ou todo o ambiente necessário para a sua execução em um Contêiner, isolando-o e permitindo que possa ser transportado para outro *host* onde haja outro *Docker* instalado, aproveitando os conceitos de computação existentes em torno de contêineres e, especificamente, no mundo Linux (*cgroups* e *namespaces*). A tecnologia do *Docker* é exclusiva porque se concentra nos requisitos dos desenvolvedores e operadores de sistemas para separar as dependências de aplicativos da infraestrutura (DOCKER 2020).

Um contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências para que o aplicativo seja executado de maneira rápida e confiável de um ambiente de computação para outro (DOCKER, 2020).

O *Docker* não pode ser confundido com virtualização, embora tenham os benefícios semelhantes, mas funcionam de maneira bem distintas. Enquanto a virtualização isola todos o ambiente do sistema operacional consumindo muito recurso do *host*, o *Docker* apenas utiliza apenas o que é necessário para a aplicação, que será isolada compartilhando o mesmo *host* com outros contêineres conforme a figura02 (DOCKER, 2020).

Figura 2 - Comparação entre Docker e Virtualização



Fonte: (DOCKER, 2020)

Utilizando o *Docker* hoje em dia, o tempo entre novas versões de um aplicativo se torna muito mais curto, mas o software em si não se torna mais simples. Pelo contrário, software e projetos aumentam em complexidade. Assim, precisamos de uma maneira de domar a besta e simplificar a cadeia de fornecimento de software (SCHENKER,2018).

Quando se utiliza-se essa tecnologia de contêiner, a redução financeira assim como os recursos tecnológicos como escalabilidade que se permite aumentar ou diminuir o consumo de memória, disco assim como o consumo de processador, e possibilitando o escalonamento vertical ou horizontal dos containers.

O isolamento de ambiente de uma aplicação em contêiner, facilita e evita os problemas mais comuns de versões, de incompatibilidade de componentes de sistemas como versão de sistema operacionais, atualizações de drives ou DLLs e bibliotecas desatualizadas.

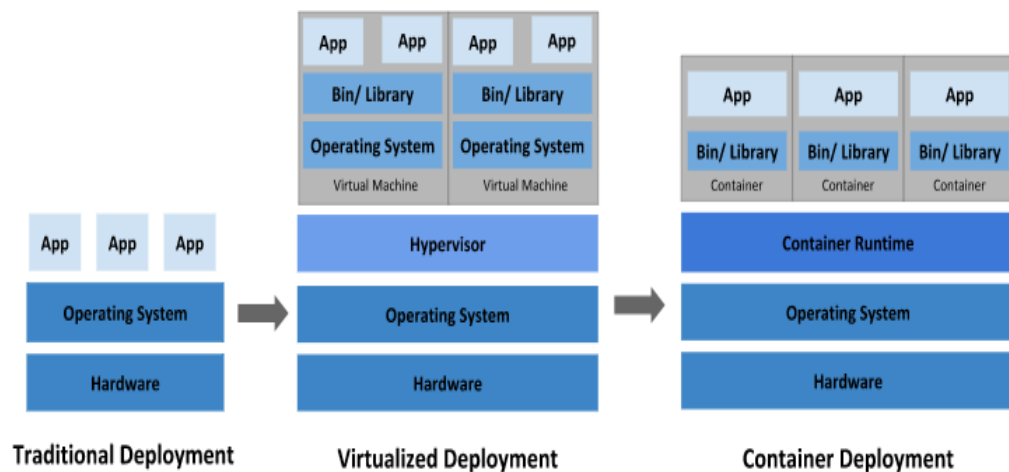
Facilidade na implantação do software feito para a tecnologia do Docker, não havendo dificuldade em sua implantação, o seu tempo de implantação é superior ou convencional.

2.2 Kubernetes

O *Kubernetes* é uma plataforma portátil, extensível e de código aberto para gerenciar cargas de trabalho e serviços em contêiner, que facilita a configuração declarativa e a automação, possui um ecossistema grande e de rápido crescimento, onde os serviços, suporte e ferramentas estão amplamente disponíveis (KUBERNETES, 2020).

O nome *Kubernetes* é originário do grego, significando timoneiro ou piloto, onde o Google abriu este projeto 2014, já vem sendo usado a uma década e meia de experiência, executando cargas de trabalho de produção em grande escala combinadas com as melhores ideias e práticas da comunidade, conforme a figura 03 abaixo (KUBERNETES, 2020).

Figura 3 - Comparativos dos três ambientes tradicional, virtualização e contêiner



Fonte: (KUBERNETES, 2020.)

Os contêineres são uma boa maneira de agrupar e executar seus aplicativos. Em um ambiente de produção, você precisa gerenciar os contêineres que executam os aplicativos e garantir que não haja tempo de inatividade, por exemplo, se um contêiner cair, outro contêiner precisa ser iniciado (KUBERNETES, 2020).

O *Kubernetes* fornece uma estrutura para executar sistemas distribuídos de forma resiliente, ele cuida do dimensionamento e do failover do seu aplicativo,

fornece padrões de implantação e muito mais. Por exemplo, ele pode gerenciar facilmente uma implantação de cenário para o seu sistema:

- a) Descoberta de serviço e balanceamento de carga
- b) Pode expor um contêiner usando o nome DNS ou usando seu próprio endereço IP. Se o tráfego para um contêiner for alto, poderá equilibrar a carga e distribuir o tráfego da rede para que a implantação seja estável.
- c) Orquestração de armazenamento
- d) Permite montar automaticamente um sistema de armazenamento de sua escolha, como armazenamentos locais, provedores de nuvem pública e muito mais.
- e) Implementações e reversões automatizadas
- f) Você pode descrever o estado desejado para seus contêineres implantados usando, e ele pode alterar o estado real para o estado desejado a uma taxa controlada. Por exemplo, você pode automatizar o *Kubernetes* para criar novos contêineres para sua implantação, remover os contêineres existentes e adotar todos os seus recursos para o novo contêiner.
- g) Empacotamento automático de lixeira você fornece ao *Kubernetes* um cluster de nós que ele pode usar para executar tarefas em contêiner, você diz o quanta CPU e memória (RAM) cada contêiner precisa, o *Kubernetes* pode ajustar contêineres nos seus nós para fazer o melhor uso de seus recursos.
- h) Autocorreção
 - i) O *Kubernetes* reinicia os contêineres que falham, substitui os contêineres, mata os contêineres que não respondem à sua verificação de integridade definida pelo usuário e não os anuncia aos clientes até que estejam prontos para servir.
 - j) Gerenciamento de segredo e configuração.
 - k) O *Kubernetes* permite armazenar e gerenciar informações confidenciais, como senhas, tokens O Auth e chaves ssh, você pode implantar e atualizar segredos e configuração de aplicativos sem reconstruir suas imagens de contêiner e sem expor segredos em sua configuração de pilha (KUBERNETES, 2020).

Além disso, *Kubernetes* opera no nível do contêiner e não no hardware, ele fornece alguns recursos geralmente aplicáveis às ofertas de PaaS, como implantação, dimensionamento, balanceamento de carga, registro e monitoramento, no entanto não é monolítico e essas soluções padrão são opcionais e conectáveis, fornecendo os alicerces para a criação de plataformas de desenvolvedores, preserva a escolha e a flexibilidade do usuário onde é importante,

Não limita os tipos de aplicativos suportados, tem como objetivo oferecer suporte a uma variedade extremamente diversificada de cargas de trabalho, incluindo cargas de trabalho sem estado, com estado e de processamento de dados. Se um aplicativo pode ser executado em um contêiner, ele deve funcionar muito bem no *Kubernetes*;

Não implanta código fonte e não cria seu aplicativo. Os fluxos de trabalho de integração, entrega e implantação contínuos (CI / CD) são determinados pelas culturas e preferências da organização, bem como pelos requisitos técnicos.

Não fornece serviços no nível do aplicativo, como middleware (por exemplo, barramentos de mensagens), estruturas de processamento de dados (por exemplo, *Spark*), bancos de dados (por exemplo, *mysql*), caches ou sistemas de armazenamento em *cluster* (por exemplo, *Ceph*) como serviços internos. Esses componentes podem ser executados no *Kubernetes* e / ou podem ser acessados por aplicativos executados no *Kubernetes* por meio de mecanismos portáteis, como o *Open Service Broker*.

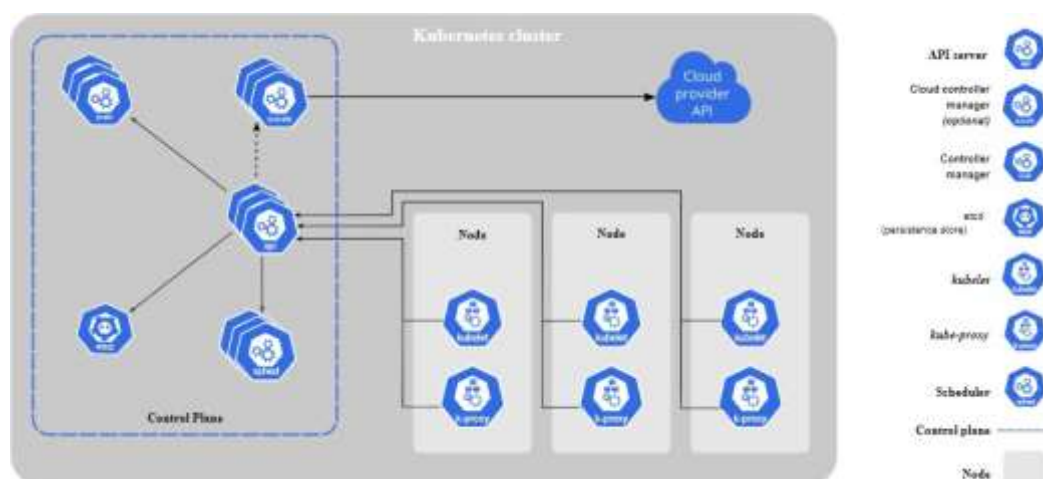
Não determina soluções de log, monitoramento ou alerta, ele fornece algumas integrações como prova de conceito e mecanismos para coletar e exportar métricas. Não fornece nem exige um idioma / sistema de configuração (por exemplo, *jsonnet*), ele fornece uma API declarativa que pode ser direcionada por formas arbitrárias de especificações declarativas. Não fornece nem adota nenhum sistema abrangente de configuração, manutenção, gerenciamento ou autocorreção da máquina.

Ademais, o *Kubernetes* não é um mero sistema de orquestração. De fato, elimina a necessidade de orquestração, a definição técnica de orquestração é a execução de um fluxo de trabalho definido: primeiro faça A, depois B e depois C. Por outro lado, o *Kubernetes* compreende um conjunto de processos de controle compostíveis e independentes que conduzem continuamente o estado atual para o estado desejado fornecido, não importa como você vai de A a C, o controle

centralizado também não é necessário. Isso resulta em um sistema que é mais fácil de usar e mais poderoso, robusto, resiliente e extensível (KUBERNETES, 2020).

Quando se implanta o *Kubernetes*, se obtém um cluster, que na verdade é que um conjunto de máquinas denominadas de Nós, que fazem a execução de aplicações em contêineres, cada grupo tem pelo menos um nó. O nó hospeda vários componentes de carga de trabalho da aplicação, o plano de controle geralmente atravessa vários computadores e um cluster normalmente é executado em vários nós, proporcionando a tolerância as falhas e permitindo alta disponibilidade.

Figura 4 – Componentes Kubernetes



Fonte: (KUBERNETES, 2020.)

Um conjunto de vários *kubernetes* e seus componentes interligados, como mostra a figura 4 acima.

- a) kube-apiserver (O servidor API é um componente dos *Kubernetes* plano de controle que expõe a API *kubernetes*)
- b) etcd (Loja de valor de chave consistente e altamente disponível usada como loja de backup da *Kubernetes* para todos os dados de cluster)
- c) kube-scheduler (Controle componente de plano que observa o recém-criado *pods* sem designado Nó, e seleciona um nó para que eles sejam executados)

- d) kube-controller-manager (Componente de controle plano que é executado Controlador de Processos)
- e) cloud-controller-manager (Um *Kubernetes* plano de controle componente que incorpora lógica de controle específica da nuvem)
- f) Node Components (Os componentes do nó são executados em cada nó, mantendo as cápsulas em execução e fornecendo o ambiente de tempo de execução de *Kubernetes*) (KUBERNETES, 2020).
 - o Kubelet (Um agente que funciona em cada Nó no aglomerado. Ele garante que Recipientes estão correndo em um *Pod*)
 - o Kube-proxy (é um proxy de rede que é executado em cada Nó em seu cluster, implementando parte dos *Kubernetes* Serviço Conceito)
 - o Container runtime (O tempo de execução do contêiner é o software responsável pela execução dos contêineres)
- g) Addons (usam recursos *Kubernetes* (DaemonSet, Implantação, etc) para implementar recursos de *cluster*)
- h) DNS (Os contêineres iniciados pela *Kubernetes* incluem automaticamente este servidor DNS em suas pesquisas de DNS)
- i) Web UI (é uma interface do usuário baseada na Web para clusters *Kubernetes*. Ele permite que os usuários gerenciem e solucionem problemas em execução no cluster, bem como no próprio cluster)
- j) Container Resource Monitoring (registra métricas genéricas de séries temporais sobre contêineres em um banco de dados central e fornece uma interface do usuário para navegar nesses dados)
- k) Cluster-level Logging (é responsável por salvar logs de contêineres em uma loja de log central com interface de pesquisa/navegação)

2.3 Contextualização

A containerização com o *Docker* possibilita o isolamento de uma app ou de toda a infraestrutura na qual ele necessita dentro de um container, e assim dessa maneira o ambiente inteiro torna-se portátil para qualquer outro host que contenha o *Docker* instalado.

Com isso reduzindo o tempo de *deploy* da aplicação, pois não há mais a necessidade de ajustes de ambiente para o seu funcionamento, o ambiente é sempre o mesmo, já foi configurado anteriormente e pode ser replicado a qualquer hora basta haver necessidade (DIEDRICH, 2015).

Justifica-se o desenvolvimento desse tema com a possibilidade da utilização do *Docker* e *Kubernetes* em todos os softwares desenvolvidos, possibilitando a sua aplicação em empresas de pequeno porte que utilizam um *Enterprise Resource Planning* (ERP) em sua rede local.

2.4 Alta Disponibilidade

No universo da tecnologia da informação (TI) o termo, Alta Disponibilidade, indica que uma determinada aplicação ou serviço esteja permanentemente disponível, 24 horas por dia, 7 dias por semana. A não tolerância as falhas, nos faz recorrer ao recurso da replicação, que é base para obtermos alta disponibilidade, muito utilizado em sistemas distribuídos, e onde as aplicações ou serviços estejam permanentemente disponíveis a quase toda maior parte do tempo.

A melhoria de desempenho utilizando a replicação é amplamente usada. Por exemplo, no armazenamento de recursos de servidores Web no cache dos navegadores e em servidores proxies Web, é uma forma de replicação, pois os dados mantidos nas caches e nos proxies são réplicas uns dos outros. O serviço de atribuição de nomes DNS mantém cópias de mapeamentos entre nomes e atributos dos computadores e é fundamental para o acesso diário a serviços pela Internet. A replicação é uma técnica para melhorar os serviços.

As motivações para a replicação são: melhorar o desempenho de um serviço, aumentar sua disponibilidade ou torná-lo tolerante a falhas (COULOURIS, 2013).

2.5 Micro Serviços

Surgido em maio 2011 em uma conferência de arquitetura de software, para desenvolvimento de softwares mais flexíveis com a possibilidade de escalabilidade e de simples manutenção do que a arquitetura tradicional a monolítica (Sistemas tradicionais, que reúnem vários módulos em só programa). Cada micro serviço é destinado uma determinada finalidade específica, assim tornando-se independente tanto na sua manutenção e implantação, mas comunicando-se com outros micros serviços através de API com recursos HTTP ou troca de mensagens, tornando-se então mais eficiente e útil a sua criação.

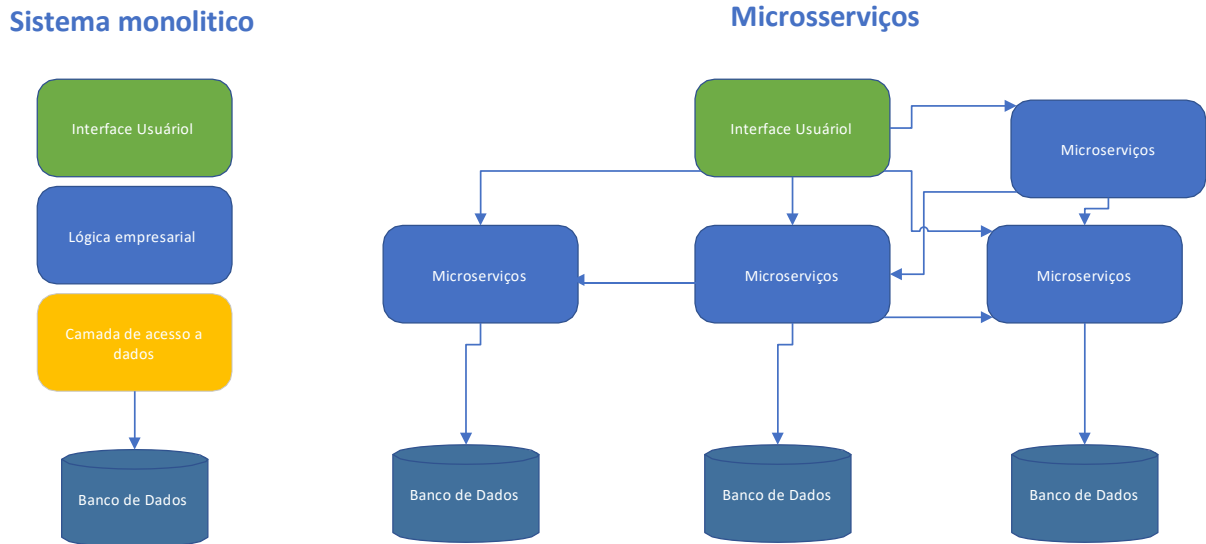
Suas vantagens na implementação de mecanismos de comunicação entre serviços e controle de falhas, além de ser relativamente pequeno, com maior facilidade no seu desenvolvimento e de melhor entendimento para o desenvolvedor.

A arquitetura de micros serviços é uma técnica que desenvolve um aplicativo único como um conjunto de pequenos serviços, cada um executando seu próprio processo e se comunicando através de mecanismos leves. Esses serviços são construídos de acordo com as necessidades de negócios e funcionam através de mecanismos de *deploy* independentes totalmente automatizados. Há o mínimo possível de gerenciamento centralizado desses serviços, que podem ser escritos em diferentes linguagens de programação e utilizam diferentes tecnologias de armazenamento de dados (LEWIS; FLOWLER, 2015).

Obviamente que nem todos os sistemas devam ser desenvolvidos nessa concepção de micro serviços, vai depender muito da necessidade e características dos tipos de aplicação que se precise utilizar para resolução dos problemas

Nesta figura 4, exemplificamos a diferença entre um sistema tradicional (monolítico) com o novo modelo do micro serviços.

Figura 5 - Comparativos do sistema monolítico e micros serviços



Fonte: DO PRÓPRIO AUTOR

Demonstrando graficamente o comportamento dos dois padrões, onde o monolítico por ser o conjunto fechado de módulos, não há como escalonar de forma individual cada módulo, já no caso dos micros serviços é fácil visualizar que cada serviço é independente um dos outros, embora se comuniquem e se relacionem entre eles, e por serem de tamanho menor e independentes fica mais fácil a sua manutenção sem interferir nos outros serviços.

3 MODELO DE APLICAÇÃO DE ALTA DISPONIBILIDADE

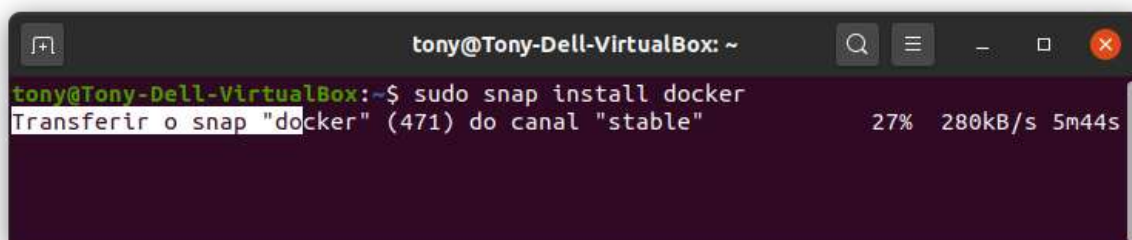
Inicialmente, devemos ter em mente o que queremos e como iremos projetar e desenvolver uma aplicação, feitas estas etapas preliminares para o desenvolvimento de um projeto de um sistema, e definindo qual rumo se quer tomar e relação a sua arquitetura se monolítico, ou se será utilizado o modelo de micros serviços, este trabalho serve como um orientador de como pode-se ou por onde começar.

Utilizaremos o software de virtualização VirtualBox da empresa Oracle, para criarmos uma **VM** com **OS** Linux Ubuntu 20.4, onde serão feitos os procedimentos necessários para nosso projeto de modelo de aplicação de alta disponibilidade, podendo ser feito também em **VPS** em Clouds ou data center que atuam no mercado, com **OS** Linux ou Windows, pós o *Docker* e *Kubernetes* estão presentes em ambas as versões, vai depender da disponibilidade financeira de recursos disponível para sua utilização.

3.1 Instalação do Docker

Para instalação do Docker, iremos utilizar uma VM com o OS Linux Ubuntu, lembrado que podemos também o fazer em versão Desktop para Windows, que também é gratuito, mas não será utilizado no nosso modelo. Digitando na linha de comando no linux, **sudo snap install docker**, para efetuar a instalação do Docker na VM, conforme figura 5 abaixo.

Figura 6 – Tela VM Linux instalando *Docker*



```
tony@Tony-Dell-VirtualBox: ~  
tony@Tony-Dell-VirtualBox:~$ sudo snap install docker  
Transferir o snap "docker" (471) do canal "stable" 27% 280kB/s 5m44s
```

Fonte: DO PRÓPRIO AUTOR

Após a instalação concluída, verifica-se a versão instalada do *Docker* na vm linux, como mostrar com o comando do *Docker* **docker version**, conforme podemos ver na figura 6 abaixo.

Figura 7 – Tela VM visualizando versão do *Docker*



```

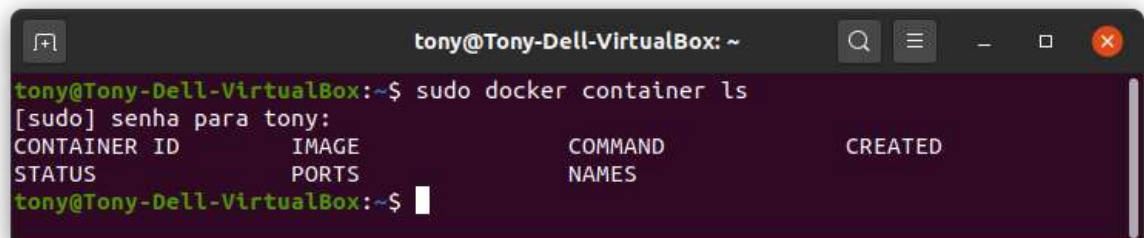
tony@Tony-Dell-VirtualBox:~$ docker version
Client:
Version:           19.03.11
API version:       1.40
Go version:        go1.13.12
Git commit:        dd360c7
Built:             Mon Jun 8 20:23:26 2020
OS/Arch:           linux/amd64
Experimental:     false
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.40/version: dial unix /var/run/docker.sock: connect: permission denied
tony@Tony-Dell-VirtualBox:~$

```

Fonte: DO PRÓPRIO AUTOR

Podendo-se visualizar a versão do *Docker* recém instalado, e que está pronto para ser utilizado. Com outro comando do *Docker* para verificamos se há alguma imagem de contêiner disponível no ambiente recém instalado, faremos isso com o comando **docker container ls** conforme figura 7 abaixo.

Figura 8 – Tela VM Linux Visualizando contêineres instalados



```

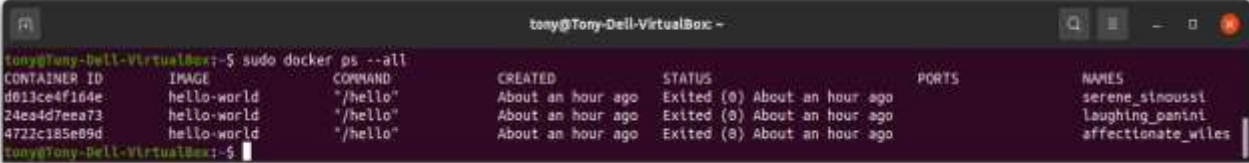
tony@Tony-Dell-VirtualBox:~$ sudo docker container ls
[sudo] senha para tony:
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
tony@Tony-Dell-VirtualBox:~$

```

Fonte: DO PRÓPRIO AUTOR

Para conhecer mais comando do *Docker*, basta utilizar o comando **docker run hello-world** que é uma imagem *Docker* para testes, iremos executar esse comando 3 vezes e logo após digitaremos outro comando *Docker* **docker ps --all** onde serão mostrados todos os containers que estão rodando conforme figura 8 abaixo.

Figura 9 – Tela VM Linux mostra os containers executando no *Docker*



```

tonyg@Tony-Deil-VirtualBox: ~$ sudo docker ps --all
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS          PORTS          NAMES
d013ce4f104e   hello-world    "/hello"                About an hour ago    Exited (0) About an hour ago          serene_sinoussi
24ea4d7eea73   hello-world    "/hello"                About an hour ago    Exited (0) About an hour ago          laughing_pantni
4722c185e09d   hello-world    "/hello"                About an hour ago    Exited (0) About an hour ago          affectionate_wiles
tonyg@Tony-Deil-VirtualBox: ~$

```

Fonte: DO PRÓPRIO AUTOR

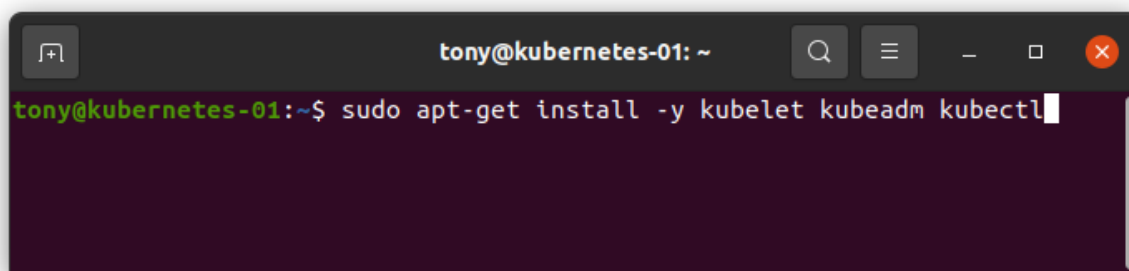
Podemos observar na figura 8, que é mostrado todos os containers que estão em execução nesse momento, na verdade só existe um container, mas três cópias do mesmo processo rodando em paralelo totalmente independente um do outro, onde cada container tem o seu identificador distinto de cada por onde o Docker faz o controle sobre o mesmo, assim se quiser pausar um container ou até mesmo mata-lo, bata informar o comando *Docker* com ação desejada com o seu **container id** para que Docker faça o que se deseja com o contêiner.

3.2 Instalação do Kubernetes

Este item demonstrara como proceder na instalação do *Kubernetes*, dependendo do ambiente ao que se quer utilizar, onde o procedimento é bem parecido com o 3.1, e também é necessário para infraestrutura do modelo de aplicação de alta disponibilidade, porque o *Kubernetes* é o orquestrador, automatizar a implantação e o dimensionamento e gerenciamento dos contêineres.

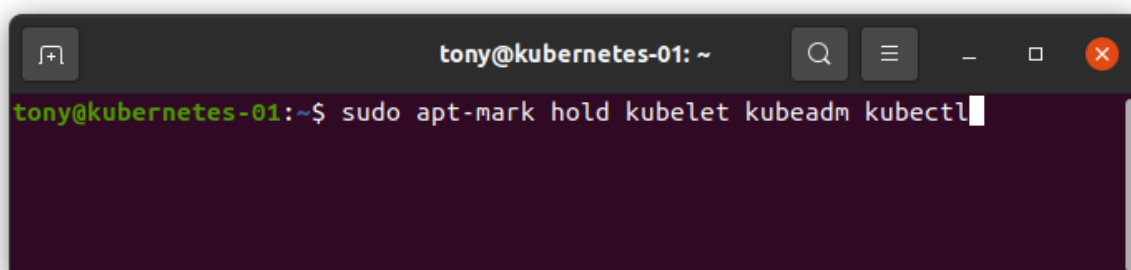
Algumas recomendações para pré-instalação verificando se cada endereço **MAC** é único para cada node(nó), verificar os adaptadores de rede e certifica-se que o modulo **iptables** está carregado.

Digitando a sequência de comandos linux para a instalação do kubeadm, kubelet, kubectl conforme figuras 10,11,12 abaixo.

Figura 10 – Tela VM Linux Instando *Kubernetes*A terminal window titled 'tony@kubernetes-01: ~' with search, menu, and window control icons. The command 'sudo apt-get install -y kubelet kubeadm kubectl' is entered and the cursor is at the end of the line.

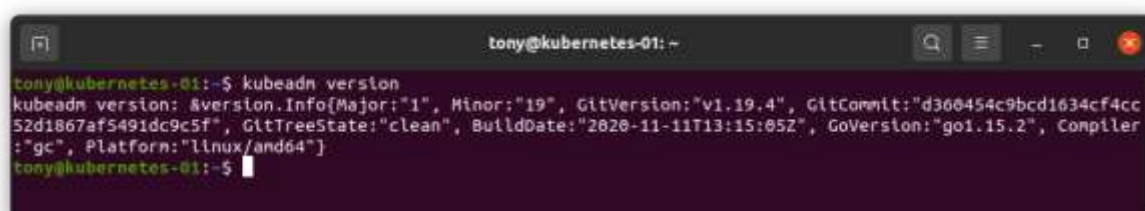
```
tony@kubernetes-01:~$ sudo apt-get install -y kubelet kubeadm kubectl
```

Fonte: DO PRÓPRIO AUTOR

Figura 11 – Tela VM Linux Instando Ferramentas *Kubernetes*A terminal window titled 'tony@kubernetes-01: ~' with search, menu, and window control icons. The command 'sudo apt-mark hold kubelet kubeadm kubectl' is entered and the cursor is at the end of the line.

```
tony@kubernetes-01:~$ sudo apt-mark hold kubelet kubeadm kubectl
```

Fonte: DO PRÓPRIO AUTOR

Figura 12 – Tela VM Linux Verificando a versão *Kubeadm*A terminal window titled 'tony@kubernetes-01: ~' with search, menu, and window control icons. The command 'kubeadm version' is entered, and the output is displayed.

```
tony@kubernetes-01:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.4", GitCommit:"d360454c9bcd1634cf4cc52d1867af5491dc9c5f", GitTreeState:"clean", BuildDate:"2020-11-11T13:15:05Z", GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
tony@kubernetes-01:~$
```

Fonte: DO PRÓPRIO AUTOR

Após efetuada a instalação das ferramentas do *Kubernetes*, podemos configurar conforme a nossa necessidade e recursos disponíveis e usufruir de suas funcionalidades, pós não basta só isolarmos a aplicação em contêiner com o Docker, é necessário a orquestração que o *Kubernetes* faz, como não permitir que não tenha inatividade de uma aplicação (monitoramento), ou mesmo escalonar os recurso de memória, disco, processador, além do balanceamento de cargas caso seja necessário para manter a alta disponibilidade da aplicação.

3.3 Escolhendo o Modelo de Aplicação

Nesta etapa iremos definir qual arquitetura de que queremos implementar na aplicação, se monolítica (sistema no modelo tradicional) ou na arquitetura de micro serviços, observando-se que para trabalhar com Docker não há restrição em relação ao tipo de arquitetura que se vai usar, só o bom senso em saber que para se fazer o escalonamento de uma aplicação monolítica requer muito mais recursos computacional que um micro serviço como já foi visto no item 2.5.

3.4 Levantamento do Requisito da Aplicação

Neste momento deve-se fazer os levantamentos básicos das informações, definindo prioridades e requisitos funcionais e não funcionais da aplicação, para iniciar o projeto de aplicação a ser feito.

3.5 Implementação da Aplicação

Na implementação do projeto de aplicação, já com tudo definido, parte para uma ferramenta de codificação que permita a containerização, preferencialmente que seja multiplataforma como por exemplo o Java ou outras que atendam a esse requisito, para poder utilização do *Docker* e obter os recursos desta tecnologia.

4 RESULTADOS E DISCUSSÕES

Os resultados deste trabalho foram avaliados em comparação da economia de recursos computacionais entre os dois modelos de aplicações monolíticas tradicionais e containerização *Docker*.

4.1 Configuração do Ambiente da Aplicação

Na configuração do ambiente quando pela primeira vez, no ambiente não *Docker* é, na maioria das vezes, até mais rápido e menos complexo, mas quando há a necessidade de se repetir essa operação, como foi visto nas seções 3.1 e 3.2, que a facilidade e rapidez de se replicar um contêiner, vemos o ganho na economia de tempo e de recursos computacionais.

4.2 Implantação e Manutenção de Aplicação

Quanto à implantação (*deploy*) de uma aplicação, é sem sombra de dúvidas que a containerização é muito mais produtiva como demonstrada nas seções 2.3 e 2.5, que relatam as vantagens significativas de se utilizar as tecnologias de contêineres, além de evitar vários problemas de incompatibilidades de bibliotecas, falhas de configurações de ambiente em desenvolvimento com o de produção.

Onde a integração infraestrutura de rede e a de software se unem evitando o desperdício de recursos de TI, onde é mais fácil a identificação de problemas, e mais rápido a sua solução.

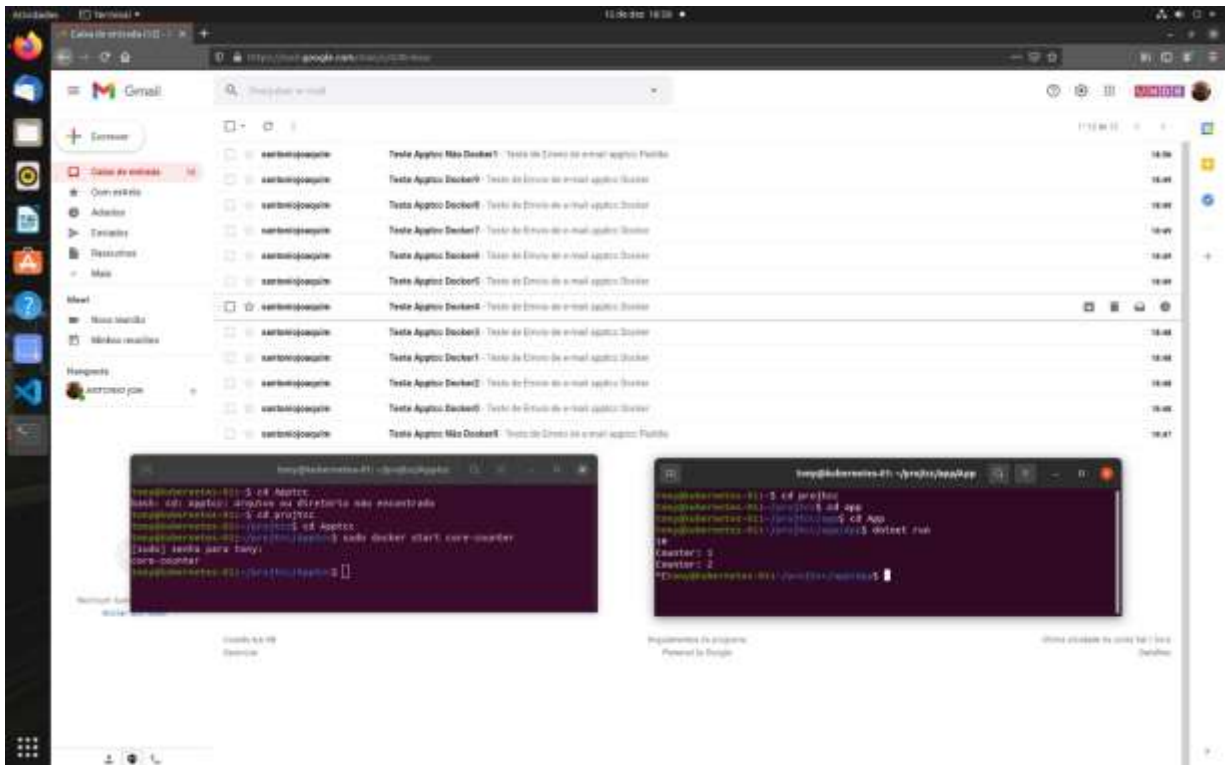
4.2 Tempo de Execução da Aplicação

Foi analisado duas aplicações feitas em C# core 3.1, ambas para ser executados em console (terminal), uma com ambientação do *Docker* e outra com a mesma funcionalidade de envio de e-mail (Correio eletrônico), no ambiente tradicional sem containerização, onde cada aplicação dispararia a quantidade de dez (10) e-mail para um determinado endereço.

Onde primeiramente foi iniciado a aplicação tradicional não *Docker* de nome (App), assim que o primeiro e-mail caiu na caixa de mensagem de correio

destino, foi executada a segunda aplicação no ambiente *Docker* de nome (*core-counter*), esse por sua vez executou todos os disparos de e-mail mesmo antes do segundo e-mail da primeira aplicação (*App*) houvesse enviado o segundo e-mail da sua sequência de dez (10) e-mails fosse concluído, conforme figura 13 abaixo.

Figura 13 – Tela VM Linux Análise das Aplicações



Fonte: DO PRÓPRIO AUTOR

Após analisar as duas aplicações em funcionamento, o monitoramento foi feito através da checagem da caixa de entrada de e-mail do destinatário, onde fica registrada por ordem de chegada e registrando a hora da chegada de cada um e-mail.

A tabela abaixo é o demonstrativo de chegada dos e-mails, destacando a diferença entre os dois primeiros e-mails enviados pelas duas aplicações (*App* e *core-counter*), e o último da aplicação da segunda aplicação *Docker* que teve o seu serviço finalizado primeiro que a aplicação tradicional, conforme demonstra a tabela 1 logo abaixo.

Tabela 1 – Comparativo de Horário de Chegada dos E-mails

Descrição	1º Email	2º Emal	3º Emal	4º Emal	5º Emal	6º Emal	7º Emal	8º Emal	9º Emal	10º Emal
Aplicação não Docker (App)	18:47	18:56								
Aplicação Docker (core-counter)	18:48	18:48	18:48	18:48	18:49	18:49	18:49	18:49	18:49	18:49
Diferença em minutos	00:01	00:08								00:07

Fonte: DO PRÓPRIO AUTOR

Nela se faz a análise dos primeiros e-mails recebido, levando-se em conta que a aplicação não *Docker* (App) foi executada primeiro e aplicação *Docker* foi iniciada só quando o primeiro e-mail do (App) que chegou na caixa de entrada às 18:47h, e só neste momento foi iniciada aplicação *Docker* (Apptcc do contêiner core-counter), e seu primeiro e-mail teve sua chegada às 18:48h, apenas um (1) minuto após a chegada do primeiro e-mail da aplicação não *Docker*, e as nove(9) e-mails restante da aplicação *Docker* tiveram sua chegada quase que imediatamente, com a variação de um (1) minuto do quarto e-mail para o 10 dessa aplicação, em quanto isso o segundo e-mail da aplicação não *Docker* foi chegar às 18:56h, com uma diferença do último (decimo) e-mail da aplicação *Docker* com a diferença de sete(7) minutos.

Isto demonstra que a aplicação no ambiente contêiner é mais independente e rápida do que aplicação em ambiente convencional.

5 CONCLUSÃO

De acordo com as pesquisas realizadas nas seções 2, 2.1, 2.2 e descritos nas seções 3, 3.1, 3.2, que deram base para este trabalho de análise das tecnologias de desenvolvimento de software.

E após análises feitas nas seções 4.1 onde faz a comparação de configuração de ambientação dos dois modelos de aplicação, demonstrando que há possibilidade de escolha depende muito do tipo de projeto a se realizar.

E na seção 4.2 que demonstra a notável e grande diferença em relação a independência de ambiente de execução do aplicativo rodando em contêiner, demonstra a diferença em relação ao envio de e-mails com grande vantagem no tempo de envio, e que pode responder à pergunta se é possível se utilizar as tecnologias *Docker* e *Kubernetes* em aplicações de pequeno porte? Não só responde, como demonstra que independe do porte da aplicação. A tecnologia de containerização ajuda muito na alta disponibilidade de aplicações, permitindo o seu acesso por muito mais tempo com tolerância a falhas bem menor que o ambiente tradicional.

5.1 Trabalhos Futuros

Propõe-se em breve a elaboração de um projeto de aplicação ERP (Estoque e Vendas Produtos), com alta disponibilidade utilizando as tecnologias *Docker* e *Kubernetes*; para fazer um comparativo prático utilizando o aplicativo com tecnologias *Docker* e *Kubernetes* e a tecnologia tradicional monolítica, através de demonstração de usabilidade e desempenho mensurando a sua escalabilidade de recursos computacionais.

REFERÊNCIAS

SCHENKER, Gabriel. **Containerize your Apps with Docker and Kubernetes: Deploy, scale, orchestrate, and manage containers with Docker and Kubernetes**. Nº. 1. UK: Packt Publishing Ltd, 2018. Nº 5.

COULOURIS, G. et al. **Sistemas Distribuidos: Conceitos e Projeto**. Nº. 5. Porto Alegre: Bookman, 2013.

VOHRA, Deepak. **Kubernetes Microservices with Docker: The Expert's voice in Open Source**. British Columbia Canada: Apress 2016.

DOCKER, Introdução. **O que é um Container**. Disponível em: <https://www.docker.com/resources/what-container>. Acesso em: 10 set. 2020.

DIEDRICH Cristiano, MUNDODOCKER, Docker. **Mas por que que o Docker é tão legal?**. Disponível em: <https://www.mundodocker.com.br/o-que-e-docker>. Acesso em: 10 set. 2020.

KUBERNETES, Containers. **O que é Kubernetes**. Disponível em: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes>. Acesso em: 10 set. 2020.

DOCKERHUB, Conceitos. **[S.l.:s.n]**, 2020. Disponível em: https://hub.docker.com/search?image_filter=official&type=image&architecture=arm64. Acesso em: 10 out. 2020.

LEWIS, James; FLOWLER, Martin. Tecnologia. **Microserviços em poucas palavras**, 2020. Disponível em: <https://www.thoughtworks.com/pt/insights/blog/microservices-nutshell>. Acesso em: 10 out. 2020.

ANEXOS

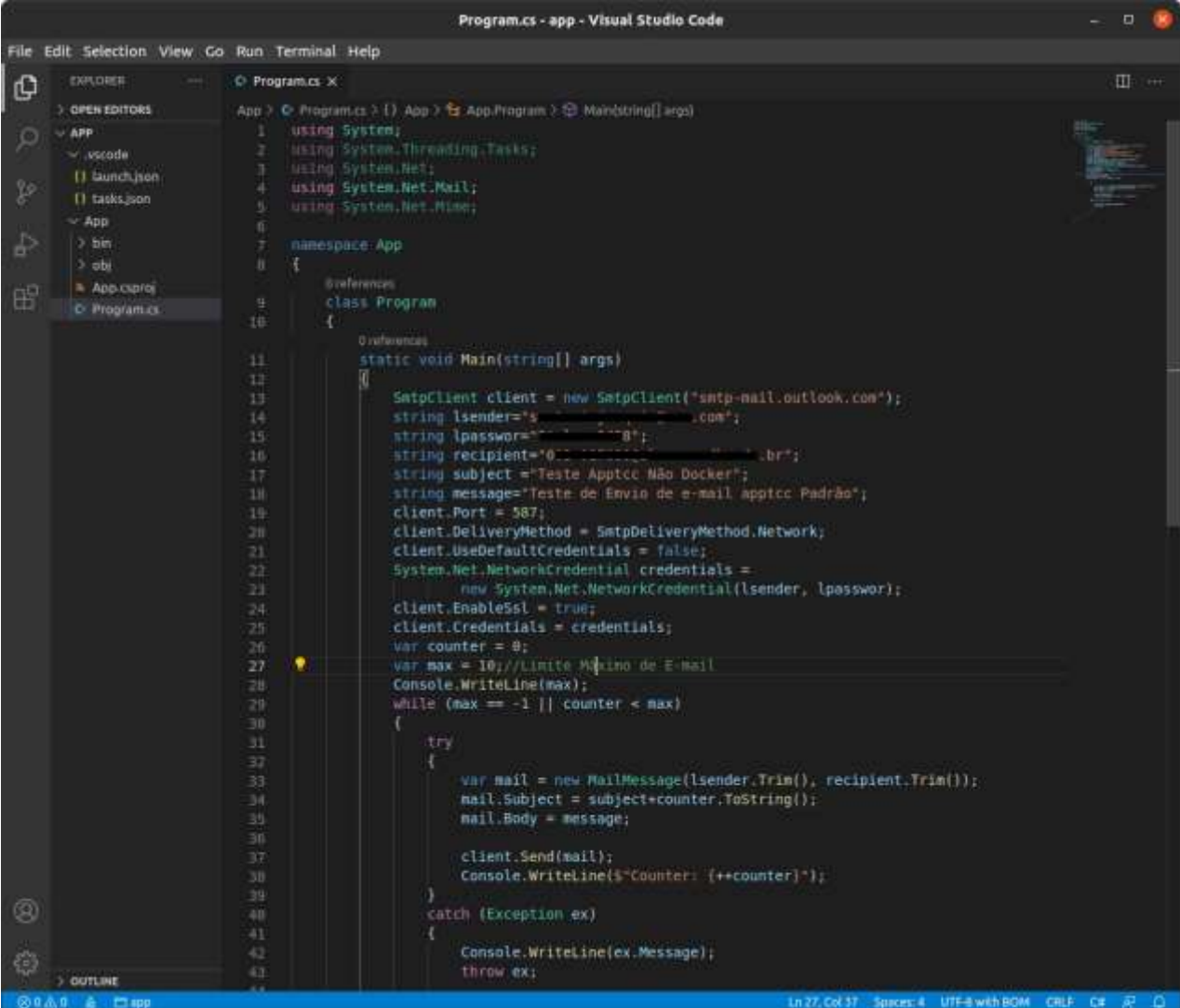
ANEXO A – Programa Fontes da Aplicação não *Docker*

ANEXO B – Programa Fontes da Aplicação *Docker* Apptcc

ANEXO C – Arquivo de definição *Dockerfile* Apptcc

ANEXO D – Arquivo do Projeto Apptcc

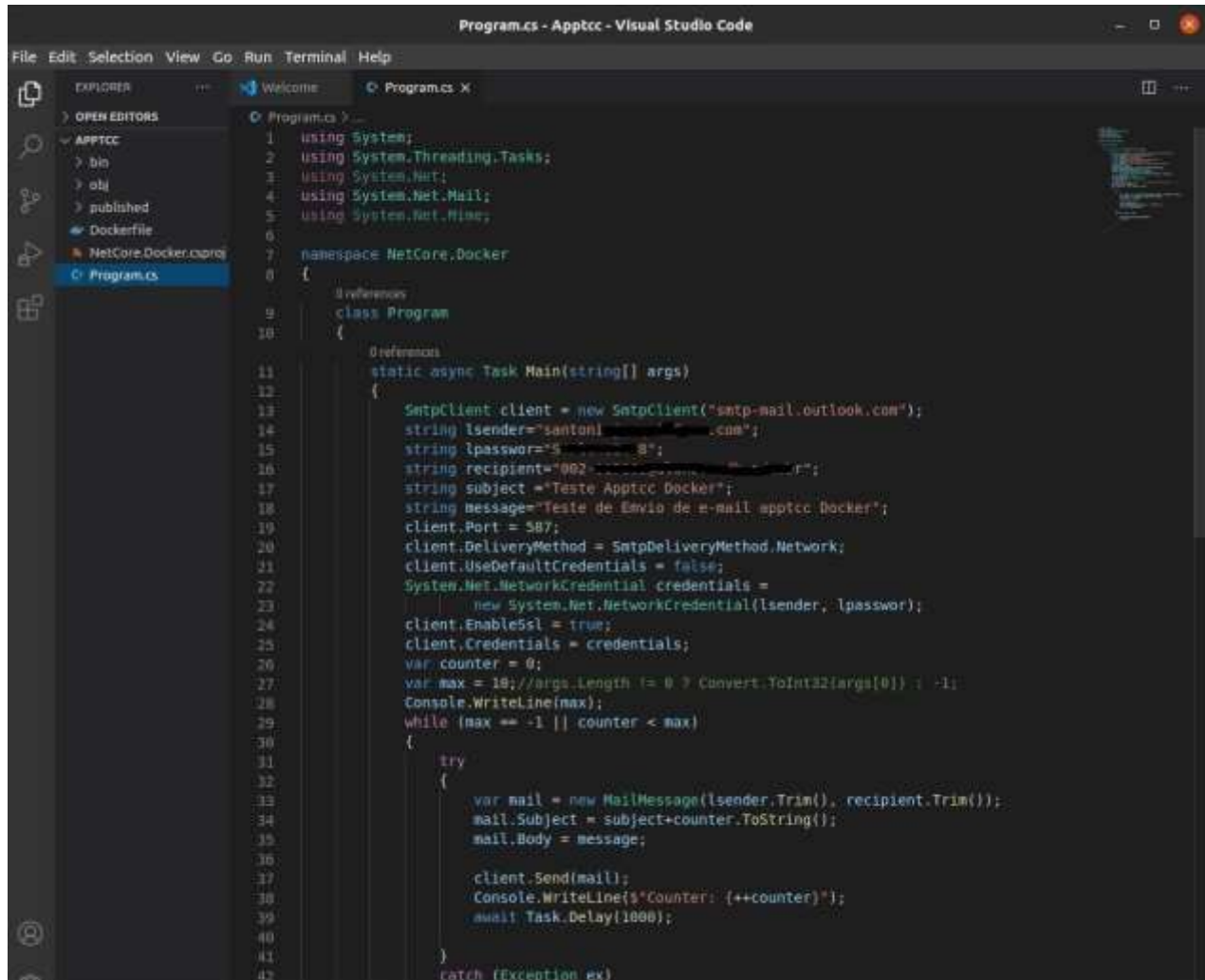
ANEXO A – Programa Fontes da Aplicação não Docker App



The image shows a screenshot of the Visual Studio Code editor with a C# file named 'Program.cs' open. The code is a console application that sends a series of emails. The code is as follows:

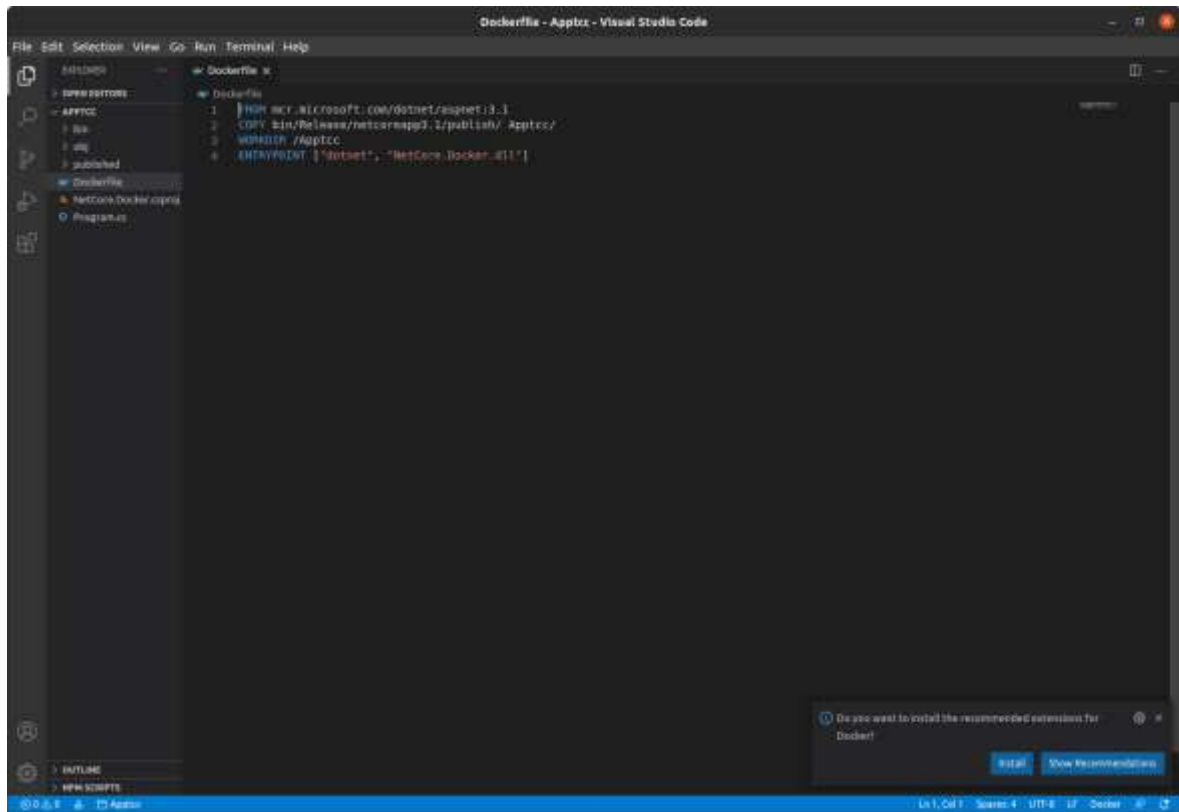
```
1 using System;
2 using System.Threading.Tasks;
3 using System.Net;
4 using System.Net.Mail;
5 using System.Net.Mime;
6
7 namespace App
8 {
9     @references
10    class Program
11    {
12        @references
13        static void Main(string[] args)
14        {
15            SmtplibClient client = new SmtplibClient("smtp-mail.outlook.com");
16            string lsender="s[REDACTED].com";
17            string lpassword="s[REDACTED]";
18            string recipient="o[REDACTED].br";
19            string subject = "Teste Apptcc Não Docker";
20            string message="Teste de Envio de e-mail apptcc Padrão";
21            client.Port = 587;
22            client.DeliveryMethod = SmtplibDeliveryMethod.Network;
23            client.UseDefaultCredentials = false;
24            System.Net.NetworkCredential credentials =
25                new System.Net.NetworkCredential(lsender, lpassword);
26            client.EnableSsl = true;
27            client.Credentials = credentials;
28            var counter = 0;
29            var max = 10; // Limite Máximo de E-mail
30            Console.WriteLine(max);
31            while (max == -1 || counter < max)
32            {
33                try
34                {
35                    var mail = new MailMessage(lsender.Trim(), recipient.Trim());
36                    mail.Subject = subject+counter.ToString();
37                    mail.Body = message;
38
39                    client.Send(mail);
40                    Console.WriteLine($"Counter: {++counter}");
41                }
42                catch (Exception ex)
43                {
44                    Console.WriteLine(ex.Message);
45                    throw ex;
46                }
47            }
48        }
49    }
50 }
```

The status bar at the bottom indicates the current cursor position: Ln 27, Col 37, Spaces: 4, UTF-8 with BOM, CRLF, C#, and a refresh icon.

ANEXO B – Programa Fontes da Aplicação *Docker Apptcc*

```
Program.cs - Apptcc - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
APPTCC
  bin
  obj
  published
  Dockerfile
  NetCore.Docker.csproj
  Program.cs
Program.cs X
Program.cs
1 using System;
2 using System.Threading.Tasks;
3 using System.Net;
4 using System.Net.Mail;
5 using System.Net.Nime;
6
7 namespace NetCore.Docker
8 {
9     class Program
10    {
11        static async Task Main(string[] args)
12        {
13            SmtplibClient client = new SmtplibClient("smtp-mail.outlook.com");
14            string lsender="santonl@outlook.com";
15            string lpassword="5...8";
16            string recipient="002...";
17            string subject = "Teste Apptcc Docker";
18            string message="Teste de Envio de e-mail apptcc Docker";
19            client.Port = 587;
20            client.DeliveryMethod = SmtplibDeliveryMethod.Network;
21            client.UseDefaultCredentials = false;
22            System.Net.NetworkCredential credentials =
23                new System.Net.NetworkCredential(lsender, lpassword);
24            client.EnableSsl = true;
25            client.Credentials = credentials;
26            var counter = 0;
27            var max = 10; //args.Length != 0 ? Convert.ToInt32(args[0]) : -1;
28            Console.WriteLine(max);
29            while (max == -1 || counter < max)
30            {
31                try
32                {
33                    var mail = new MailMessage(lsender.Trim(), recipient.Trim());
34                    mail.Subject = subject+counter.ToString();
35                    mail.Body = message;
36
37                    client.Send(mail);
38                    Console.WriteLine($"Counter: {++counter}");
39                    await Task.Delay(1000);
40                }
41            }
42            catch (Exception ex)
```

ANEXO C – Arquivo de definição *Dockerfile* Apptcc

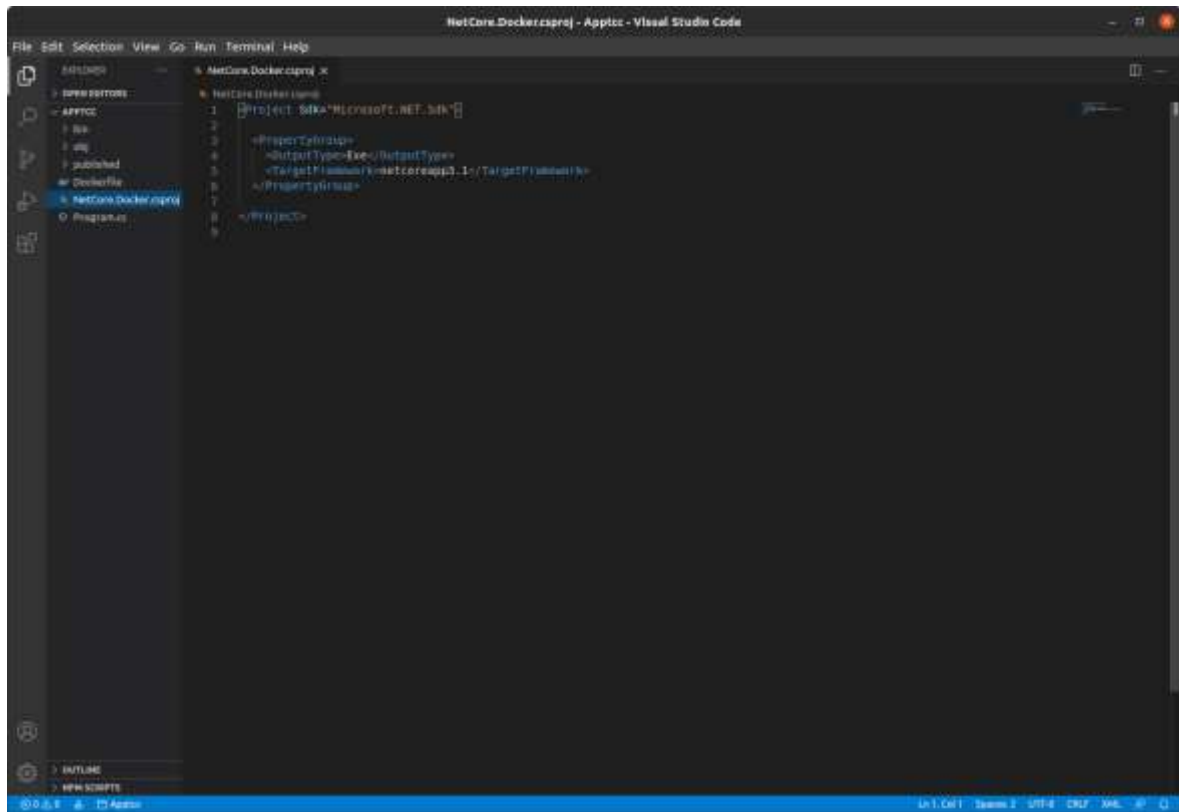


The image shows a screenshot of the Visual Studio Code editor with a Dockerfile open. The Dockerfile contains the following instructions:

```
1 FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
2 COPY bin/Release/netcoreapp3.1/publish/ Apptcc/
3 WORKDIR /Apptcc
4 ENTRYPOINT ["dotnet", "NetCore.Docker.dll"]
```

The interface includes a sidebar on the left with a file explorer showing folders like 'Apptcc' and 'published'. A notification in the bottom right corner asks if the user wants to install the recommended Docker extension.

ANEXO D – Arquivo do Projeto Apptcc



The image shows a screenshot of the Visual Studio Code editor. The title bar reads "NetCore.Docker.csproj - Apptcc - Visual Studio Code". The interface includes a menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help), a sidebar with a file explorer, and a main editor area. The file explorer shows a project structure with folders like "src" and "published", and files like "Dockerfile" and "Program.cs". The "Dockerfile" file is selected and its content is displayed in the editor. The Dockerfile content is as follows:

```
1 FROM microsoft/dotnet:2.1-sdk AS build
2
3 WORKDIR /project
4
5 COPY . .
6
7 RUN dotnet build -c Release -o /out
8
9 FROM microsoft/dotnet:2.1-runtime AS runtime
10
11 WORKDIR /project
12
13 COPY --from=build /out .
14
15 ENTRYPOINT ["dotnet", "Program.dll"]
```