Estratégias para zero-downtime releases: um estudo comparativo

Diego Quirino Silva;
Orientador: Everton Gomede
Departamento de Computação
Universidade Estadual de Londrina
E-mails: diego.quirinos88@gmail.com, evertongomede@uel.br

Resumo: A adoção da entrega contínua impõem novos desafios, aqueles que desejam sua aplicação, tem como um dos seus princípios, fazer constantes implantações de novas versões ou correções de *bug*, no entanto tal ação pode, quando não planejado incorrer na indisponibilidade temporária do sistema ou serviço, observando essa situação é necessário apresentar abordagens de como reduzir ou até mesmo zerar esse risco. Por fim através desse artigo é apresentado um comparativo das possíveis estratégias para tal e as vantagens e desavantagens delas, também é descrito uma possível abordagem, que conclui-se que pode ser satisfatório para o problema de indisponibilidade.

Palavras-chave: *downtime* zero, entrega contínua, virtualização.

1 INTRODUÇÃO

Qualquer processo de desenvolvimento de *software* tem várias etapas, mas uma das mais críticas é o momento da entrega, seja a de um novo produto, evolução ou correção de um *bug*, é a partir desse momento que verdadeiramente o *software* começa entregar valor para seu cliente.

A entrega de *software* deve ser algo previamente planejado. Identificando assim qualquer problema que possa ocorrer na hora da implantação. O planejamento quando bem executado elimina uma variedade de problemas principalmente aqueles que podem ser identificados nas fases de testes, como unitários, de aceitação e desempenho.

Porém junto entrega contínua, que é capacidade de entregar software a qualquer momento de forma confiável e com qualidade [1] impõem novos desafios. Entre eles está o momento em pôr a nova aplicação ou atualização para o acesso dos clientes, este normalmente é um momento delicado onde algumas coisas podem dar erradas, mesmo com

conjuntos de teste nas etapas anteriores e deixar a aplicação indisponível por minutos ou por horas (pior cenário).

Em um ambiente de entrega contínua [1] onde uma aplicação pode ser implantada várias vezes ao dia, seria inaceitável que ela corresse o risco de ficar indisponível o mesmo número vezes.

Essa indisponibilidade é conhecida como *downtime*. Um sistema ou serviço indisponível por qualquer que seja o motivo afetará a organização e seus clientes, em ambientes de operações de risco isso só é aceitável dentro do SLA [1], e pode de trazer prejuízos [2][3]. Segundo [1] o zero *downtime* na entrega contínua é capacidade de mover os usuários entre as versões de modo quase instantâneo.

O artigo pretende apresentar um estudo comparativo entre as estratégias que mostrem um *downtime* de migração com menor tempo possível ou até mesmo que seja igual a zero, esta transição entre a versão antiga e a nova deve ser com menor impacto possível e imperceptível aos usuários, pois toda vez que são afetados por uma indisponibilidade o grau de confiança diminui até que o cliente desista do uso do sistema.

Existem algumas maneiras de enfrentar esse problema sejam mitigando-o ou eliminando-o, as possíveis estratégias: *blue-green deployment*, *canary release* ou em último caso *rollback* [1].

Além das estratégias acima, automatizar o processo de gerenciamento de configuração é essencial. Além de integrar e implantar de forma automatizada a configuração do ambiente também deve seguir os mesmos pressupostos.

Adoção de uma ou mais estratégias pode fornecer o diferencial competitivo essencial para as organizações nos dias atuais.

Este trabalho está divido da seguinte forma: na Seção 2 é feita fundamentação para demonstrar os conceitos envolvidos, na Seção 3 são descritos as abordagens e as possíveis estratégias a serem aplicadas, na Seção 4 são demonstrados os resultados dos comparativos e as limitações envolvidas nas estratégias e, por fim, na Seção 5 é apresentada a conclusão e a proposta para trabalhos futuros.

2 FUNDAMENTAÇÃO E TRABALHOS RELACIONADOS

A entrega contínua é a capacidade de diminuir a transformação de uma ideia seja ela interna ou externa em valor, ou em outras palavras *software* funcionando em menor tempo possível [4].

Entrega contínua traz consigo muitos benefícios que são visíveis e concretos. Em [5][6] mostra-se que mesmo em organizações que já possuíam toda uma estrutura adequada de processo de *software* a entrega contínua permitiu a eles evoluí-lo ainda mais.

Alcançar a entrega contínua vai além da execução das práticas sugeridas por essa técnica, é necessário um esforço em direção em planejar e projetar a arquitetura, e quais serão custos envolvidos. Com o devido projeto estabelecido, permite-se que o ambiente seja propício para implantação contínua, e este deve ser um atributo de qualidade [7]. Preparar uma tática a fim de alcançar esse atributo é defendido por [7] aponta que a tática para isso consiste em permitir a integração contínua, habilitar automação de testes, habilitar implantação rápida e robusta operações e permitir ambientes flexíveis e sincronizáveis.

Fontes do *downtime*: falta de controle sobre as mudanças realizadas em ambiente de produção [1], *software aging* que ao longo do tempo faz que haja degradação da aplicação causando indisponibilidade [3][8][9].

Malefícios trazidos pelo *downtime*, em uma pesquisa qualitativa de [2] traz os seguintes resultados dos impactos causados, 50% na interrupção dos negócios e serviços de TI, 37% na redução de produtividade, 34% impacto negativo nas organizações de TI e 24% impacto negativo nos negócios.

A replicação de *hardware* e/ou *software* é umas das técnicas aplicadas quando se trata de evitar o *downtime* em ambientes de alta disponibilidade, a forma como é realizada diverge entre autores [10][3][8][9], porém a replicação é o mecanismo utilizado.

Ao se escolher a estratégia para mitigar o *downtime* deve-se levar em consideração o TCO (Total Cost Ownership), a fim de orientar a melhor solução. Essa é métrica que considera todos os custos envolvidos com recursos de *software* e infraestrutura da organização, da sua aquisição até seu descarte [11][12] [13].

MTTR (Mean Time to Repair) é outro indicador que é atribuído ao tempo o qual leva para repara algo, que pode ser um erro ou uma indisponibilidade de um serviço. É fornecido pela divisão do número total de horas gerado por falhas dividido pelo total de falhas. Ex: MTTR = 180 min / 6 falhas = 30 minutos [14].

O SLA (Service Level Agreement) é utilizado para determinar em contrato qual será a obrigatoriedade o qual um serviço deverá permanecer disponível. Este é utilizado

em *clouding computing*, mas não se restringe a ele apenas, pode estar associado a serviços de telecomunicação, serviços *web* ou qualquer outro o qual o cliente precise de uma garantia de funcionamento dentre de um período de tempo [15][16].

Os indicadores TCO, MTTR e o SLA são de extrema importância, pois através deles pode se orientar a estratégia a ser adotada e verificar se os resultados correspondem ao que era esperado inicialmente.

2.1 ITIL

As organizações para serem bem-sucedidas devem direcionar seus esforços e recursos numa mesma direção para otimizar os resultados, no entanto em algumas organizações isso não é realizado dessa forma, destacando-se de forma negativa principalmente a área de TI que é subestimada e mal compreendida, e vista mais como uma despesa e não uma atividade vital do negócio e geradora de lucro.

Isso se deve porque existe um aparente distanciamento entre a área de TI e os objetivos organizacionais, porque muitas vezes a área de TI não consegue comunicar de forma efetiva os benefícios da adoção de uma dada tecnologia ou a implementação de uma nova abordagem para um problema existente ou uma oportunidade de mercado, deixando espaços para má interpretação, ou falta de entendimento daqueles que vão fornecer o comprometimento necessário para execução dos novos planos.

Observando esse cenário, a OGC (Office of Governement Commerce) na década de 80 produziu a primeira versão do ITIL, e que foi evoluindo durante os anos seguintes e que em 2007 foi atualizado para versão 3 [17].

O ITIL (Information Technology Infrastruture Library) é um framework, ostensivo, o qual apresenta um conjunto de diretrizes para projetar, implementar e manter serviços de TI. Este documento a partir da versão 3 apresenta 5 livros que são o core e cobrem todo o ciclo de vida de um serviço [17]. Os livros são divididos da seguinte forma estratégia de serviço, desenho de serviço, transição de serviço, operação de serviço, melhoria de serviço continuada.

A implementação do ITIL dentro da organização faz com que haja um alinhamento entre a área de TI e os objetivos estratégicos organizacionais [17], de modo que favoreça os resultados de mais alto nível. No entanto, cabe enfatizar que esse não é um processo fácil pois apesar de fornecer as diretrizes necessárias, não são apresentadas soluções concretas cabendo a cada organização implementá-las.

2.1.1 Transição de Serviços

De todos os livros do *framework* ITIL um que melhora o compreendimento deste artigo é a transição de serviço. Organizações procuram atender da melhor forma possível o mercado, e isso significa transformações constantes de oportunidades em ganhos e ao mesmo tempo não oferecer riscos aos usuários já estabelecidos. Assim é necessário estabelecer os procedimentos básicos para que tanto a oportunidade quanto a diminuição de riscos decorram de forma segura e eficaz.

O processo de transição de serviço cobre de forma precisa os aspectos da mudança no status de desenvolvimento para a entrega [18], isso é de tal forma importante que a maioria dos software são considerados serviços e alguns são essências e que não devem parar de forma alguma. Além disso a transição de serviço combina elementos como gerenciamento de liberação, gerenciamento de programas e riscos colocando-os sob o contexto de gerenciamento de serviço.

Com o emprego da entrega contínua, a evolução passa a ser constante assim como a transição de versões de um serviço, sendo necessário adotar processos para que o emprego dessa técnica seja benéfico.

Os processos básicos necessários para a transição de serviço, que estão descritos no *framework* ITIL são:

Planejamento de transição e suporte, o objetivo desse processo é planejar e coordenar os recursos para que a nova função esteja dentro do que foi pré estabelecido em termos de custo, qualidade e tempo; garantir padronização de um *framework* reusável e prover planos compreensivos de forma a ficar claro os planos de transição de serviço [19].

Gerenciamento de mudança, tem como objetivo garantir que as mudanças sejam registradas e validadas, autorizada, priorizada, planejada, testada, implementada, documentada e revista a fim de serem controladas [19].

Gerenciamento de configuração e ativos de serviço, possui o objetivo de definir e controlar os componentes do serviço de infraestrutura e manter de forma precisa as informações de configuração [19].

Gerenciamento de implantação e liberação tem o objetivo de permitir que o planejamento de implantação e liberação seja de fácil compreensão para que tantos os clientes quanto negócios possam se alinhar a tais mudanças; alcançar de forma bem-sucedida a construção, instalação e teste no ambiente alvo no prazo estabelecido; a mudança introduzida

é capaz de atender os requisitos estabelecidos; existe um impacto mínimo serviço, operação e apoio a organização; clientes, usuários e a equipe de gerenciamento de serviço estão satisfeitos com a transição de serviço [19].

Teste e validação de serviços tem o objetivo de garantir confiança no processo de liberação de forma que os novos serviços atinjam os resultados e que atenda os valores projetados de custo, capacidade e restrições; validar que o serviço entregue esteja dentro dos níveis de desempenho esperados; validar que o serviço esteja dentro do esperado na especificação das condições dos usuários; confirmar que os requisitos definidos por usuários e *stakeholders* estejam corretos a fim de evitar correções de serviços já em produção [19].

Avaliação tem como objetivo de garantir que as mudanças pretendidas no serviço estejam de acordo com a capacidade, recursos e restrições organizacionais fornecidos; fornecer bons resultados para que o gerenciamento de mudança possa tomar a decisão de aprovar a mudança efetivando-a [19].

3 ESTRATÉGIAS DE ZERO-*DOWNTIME*

Para muitas organizações o *software* é um serviço e sua razão de existir, como por exemplo: Netflix®, Google Doc®, *Home Broker* de instituições financeiras. Essas organizações não podem conviver com indisponibilidade, pois isso acarreta desde insatisfação dos clientes até processos financeiros que podem levá-los ao seu fim.

Enquanto as falhas ocorrem em ambientes controlados como de desenvolvimento, homologação não há preocupação, no entanto quando isso afeta o cliente é um transtorno muito grande e deve ser reduzido ou eliminado, pois tais transtornos reduzem a confiança do usuário.

Grande parte dos usuários dificilmente aceitam indisponibilidade de algum serviço, e isso em alguns casos justifica-se, pois alguns serviços são essências, como por exemplo os financeiros ou de saúde.

A fim de mitigar ou eliminar essa indisponibilidade existem estratégias como a *blue-green deployment* e a *canary release* que podem ser implementadas.

3.1 IMPLANTAÇÃO *BLUE-GREEN*

A estratégia *blue-green* funciona com o conceito de comutação entre ambientes de produção, denominado *blue* e *green*, onde a nova versão é implantada em um desses ambiente e as requisições dos usuários são direcionados para lá [1], como pode ser visto na Figura 1.

Na etapa de implantação da nova versão se algum problema ocorrer em nada afetará os clientes, pois os mesmos estarão em um ambiente diferente e isolados de qualquer efeito da nova versão. Dessa forma o serviço não é interrompido, e permite a verificação de qual foi à fonte do problema para sua correção.

Quando a implantação ocorre de maneira esperada e não apresenta nenhum empecilho, então os clientes são comutados para a nova versão através da configuração do roteador, e dessa forma passam a utilizá-la.

É evidente que alguns problemas apenas são perceptíveis quando há uma carga real, assim sendo, se após a comutação dos clientes para o ambiente *blue* for reportada alguma instabilidade ou degradação do serviço, o processo de chaveamento é feito novamente para versão antiga, sem que ocorra indisponibilidade completa da aplicação.

Mesmo em organizações com poucos recursos isso pode ser realizado simplesmente configurando para que a aplicação possa ser executada de forma paralela e independente no mesmo servidor usando tecnologias de virtualização, não se exigindo um ambiente distribuído real. Organizações que possuem mais recursos podem usar a mesma estratégia com uso de servidores reais distribuídos.

Deve-se mencionar também que a questão do banco de dados usado pela aplicação pode ser fonte da paralisação ou instabilidade do serviço e precauções devem ser tomados. A realização disso é primeiramente distinguir o motivo da nova implantação se é de atualização de aplicação ou mudança de dados.

Atualização de aplicação, assumindo que o banco de dados seja a mesma para qualquer versão da aplicação que estiver sendo executada em nada afetará os clientes, porém quando se tratar de mudança no esquema de dados, essas mudanças devem ser realizadas antes da aplicação começar a interagir com ela através da refatoração do banco de dados que dê suporte tanto a versão antiga como a nova [17].

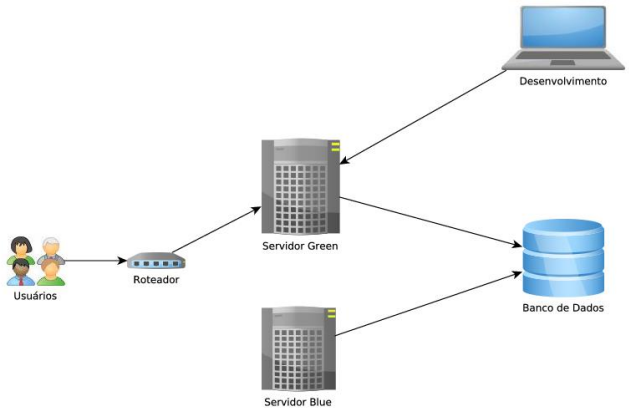

Figura 1 - *Blue-green* arquitetura

3.2 *CANARY RELEASE*

Canary release é uma técnica na qual apenas um pequeno grupo pré-selecionado tenha acesso à nova versão, como visto na Figura 2, dessa forma diminuindo os riscos associados a implantação [1][18].

Manter apenas uma versão da aplicação traz benefícios como melhor gerenciamento de correção de *bugs* e infraestrutura, mas isso traz um lado negativo. Esse lado é percebido na forma com que os *bugs* aparecem apenas quando a aplicação está em ambiente de produção [1].

Essa técnica permite zero *downtime* e expõe apenas um pequeno grupo a possíveis problemas.

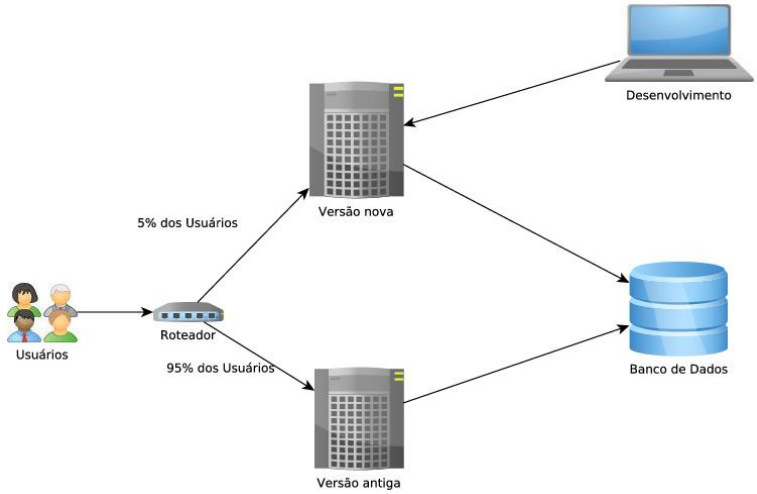

Figura 2 - *Canary release* arquitetura

Os benefícios associados ao *canary release* é que a qualquer sinal de problemas seja funcionamento inadequado de uma nova funcionalidade ou desempenho abaixo do esperado, estes afetarão apenas um pequeno grupo mais tolerante a esses problemas, diferente da estratégia anterior onde todos os usuários podem ser impactados. Há também a possibilidade de pontuar as funções mais utilizadas e descontinuar aquelas que não são usadas, essa técnica é conhecida como teste A/B. Essa técnica libera os desenvolvedores de manter funções inúteis para trabalhar em funcionalidades mais importantes [1].

Esta estratégia possui um maior grau de trabalho referente à arquitetura usada, pois se exige configurações específicas para que apenas um grupo tenha acesso à nova funcionalidade.

3.3 VIRTUALIZAÇÃO LEVE (*CONTAINER*)

As estratégias acima necessitam de meios de implementação, uma das formas na qual pode ser realizada é através da virtualização. A virtualização já é a tecnologia usada nos principais serviços de *cloud computing*, pois consegue o melhor aproveitamento dos recursos disponíveis. Dessa forma cabe ressaltar uma nova forma de virtualização conhecida como virtualização leve ou *container*.

Essa tecnologia foi desenvolvida para ser uma espécie de virtualização, mas com enfoque em leveza e velocidade. Da mesma forma que a virtualização convencional, ela tende a gerar um TCO menor para seus usuários, a grande diferença entre as técnicas é que o

container não utiliza o monitor de máquina virtual, *kernell* de SO redundante e bibliotecas, tornando assim mais leve [19].

A tecnologia de *container* se consolidou através do desenvolvimento de versões especiais do Linux, um *container* é alocação de uma porção dos recursos como CPU, entrada e saída de rede, memória RAM da máquina hospedeira. Alguns dos produtos que estão disponíveis são: LxC, Docker, Linux V-Server entre outros [19].

Essa tecnologia possui ainda alguns modelos de aplicação, cabe ressaltar que uns dos modelos são especificamente desenhados para as estratégias de entrega, esses possuem uma interface que se comunica em nível de serviço com os *containers*. A capacidade de isolação entre os vários *containers* ajuda em testes e desenvolvimento e pode ser usado em conjunto ao processo de integração contínua [19].

Apesar das vantagens, essa tecnologia possui algumas limitações que são apontadas pelo autor [20], como a falta de interoperabilidade em casos de diferença entre SO do *container* e SO do hospedeiro e problemas de segurança já que o *kernell* do sistema é o mesmo utilizado por todos os *containers*.

4 ANÁLISE E RESULTADOS

As possíveis abordagens para mitigação de *downtime* podem ser elencadas nas seguintes estratégias: *Blue-Green* e container [19], *Blue-Green* e virtualização [3][8][9], *Canary* e *container* e por fim *Canary* virtualização. Tanta a estratégia *blue-green* quanto a *canary* release podem esbarrar na questão de infraestrutura distribuída o que exigiria da organização que desejasse melhorar sua disponibilidade um aumento de investimento nos recursos, e é essa pode ser uma opção inviável.

Nem sempre a organização dispõe de servidores ociosos, devido ao custo, de forma a aplicar estratégia *blue-green* distribuída, e por outro lado pode haver servidores que não estejam consumindo todos os recursos de *hardware* disponível. Assim com tecnologias de virtualização ou *container* ambos os problemas seriam evitados de uma única vez.

A virtualização é uma técnica testada e suportada e está associada com ambientes que possuem serviços de alta disponibilidade. Virtualização fornece a capacidade de apenas um equipamento como, por exemplo, um servidor de criar vários recursos virtuais [8][21] permitindo assim a organização executar várias tarefas em um unico servidor, um

exemplo disso seria a execução de várias instâncias de servidores *web*, onde cada qual tem sua própria rede mas compartilhando a memória RAM do servidor [22].

Na Tabela 1 é mostrada a comparação de tempo em relação a algumas operações comuns aos sistemas virtualizados.

Tabela 1 - Desempenho dos tipos de virtualização (Fonte: [19])

	Pôr em operação	*Deploy* manual	*Deploy* automatizado	Tempo de *boot*
Bare Metal (Sem MMV)	Dias	Horas	Minutos	Minutos
Virtrualização	Minutos	Minutos	Segundos	< 1 minuto
Virtualização Leve (*Container*)	Segundos	Minutos	Segundos	Segundos

Blue-Green e virtualização leve (*container*)

Vantagens: baixo custo, pois não necessita da aquisição de novos equipamentos; Melhor desempenho, como a tecnologia de *container* possui uma camada a menos ele tem um melhor processamento [22].

Desvantagens: alta complexidade na instalação e gerenciamento da virtualização leve, pois é uma tecnologia nova [22]; Mão de obra especializada, precisa de pessoas com alto conhecimento na implantação e gerenciamento de ambientes virtualizados; Pode causar impacto para alguns usuários caso o novo *deploy* apresentar problemas em novas funcionalidades já que essas não existem na antiga versão.

Blue-Green e virtualização

Vantagens: fácil implementação, precisa apenas de replicação do ambiente de produção e automação para comutá-los a cada novo *deploy* ou estabelecer permissões para um responsável que possa realizar a ação; Baixo custo, não necessita da aquisição de novos equipamentos; Menor complexidade de instalação em relação ao *container*.

Desvantagens: mão de obra especializada, precisa de pessoas com alto conhecimento na implantação e gerenciamento de ambientes virtualizados; Pode causar impacto para alguns usuários caso o novo *deploy* apresentar problemas.

Canary e virtualização leve (*container*)

Vantagens: baixo impacto aos usuários já que apenas um pequeno grupo mais tolerante será afetado; Baixo custo, não necessita da aquisição de novos equipamentos;

Desvantagens: mão de obra especializada, precisa de pessoas com alto conhecimento na implantação e gerenciamento de ambientes virtualizados; Alta

complexidade, pois necessita tanto configurar a tecnologia de virtualização leve quanto configurações de rede a fim de separar os usuários que terão acesso à nova versão.

Canary e virtualização

Vantagens: baixo impacto aos usuários já que apenas um pequeno grupo mais tolerante será afetado; Baixo custo: não necessita da aquisição de novos equipamentos; Melhor desempenho, pois a tecnologia de *container* possui uma camada a menos de *software*.

Desvantagens: complexidade média, pois precisa de uma maior configuração de rede para que apenas um grupo tenha acesso à nova versão.

A Tabela 2 e Tabela 3 apresentam de forma direta as vantagens e desvantagem de cada uma das combinações, logo abaixo é apresentada de forma mais detalhada cada um dos pontos das duas tabelas.

Tabela 2 - Vantagens das abordagens combinadas

	BG & V[1]	BG & VL[2]	C & V[3]	C & VL[4]
Baixo custo	●	●	●	●
Transparente para usuário			●	●
Baixa complexidade	●			
Desempenho		●		●

Tabela 3 - Desvantagens das abordagens combinadas

	BG & V[1]	BG & VL[2]	C & V[3]	C & VL[4]
Mão de obra especializada	●	●	●	●
Alta complexidade		●	●	●
Impacto aos usuários	●	●		●

Detalhe de cada item da Tabela 2 e Tabela 3.

Vantagens:

1. Baixo custo, pois não necessita da aquisição de novos equipamentos em todas as combinações;

2. Melhor desempenho no caso BG/C & VL, como a tecnologia de *container* possui uma camada a menos ele tem um melhor processamento [22].

3. Fácil implementação no caso BG/C & V, precisa apenas de replicação do ambiente de produção e automação para comutá-los a cada novo *deploy*.

[1] *Blue-Green* e Virtualização
[2] *Blue-Green* e Virtualização Leve
[3] *Canary release* e Virtualização
[4] *Canary release* e Virtualização Leve

4. Menor complexidade no caso BG/C & V de instalação em relação ao *container*, pois a tecnologia de virtualização leve a algo recente [22] em comparação com a virtualização padrão.

5. Baixo impacto aos usuários no caso C & V/VL, já que apenas um pequeno grupo mais tolerante será afetado;

Desvantagens:

1. Alta complexidade no caso BG/C & VL, pois instalação e gerenciamento da virtualização leve é uma tecnologia nova [22];

2. Mão de obra especializada, precisa de pessoas com alto conhecimento em todos os casos por causa do uso da virtualização.

3. No caso BG & V/VL Pode causar impacto para alguns usuários caso o novo *deploy* apresente problemas em novas funcionalidades já que essas não existem na antiga versão.

4. Alta complexidade no caso C & V/VL, pois necessita de maior configuração de rede para que apenas um grupo tenha acesso, além da complexidade da própria virtualização leve.

Tais abordagens são teoricamente eficazes porque a virtualização traz uma série de benefícios e pode ser combinada com a última fase da entrega contínua.

A Figura 3 demonstra as etapas de execução após o *commit* da nova funcionalidade no sistema de controle de versão usando uma das possíveis abordagens:

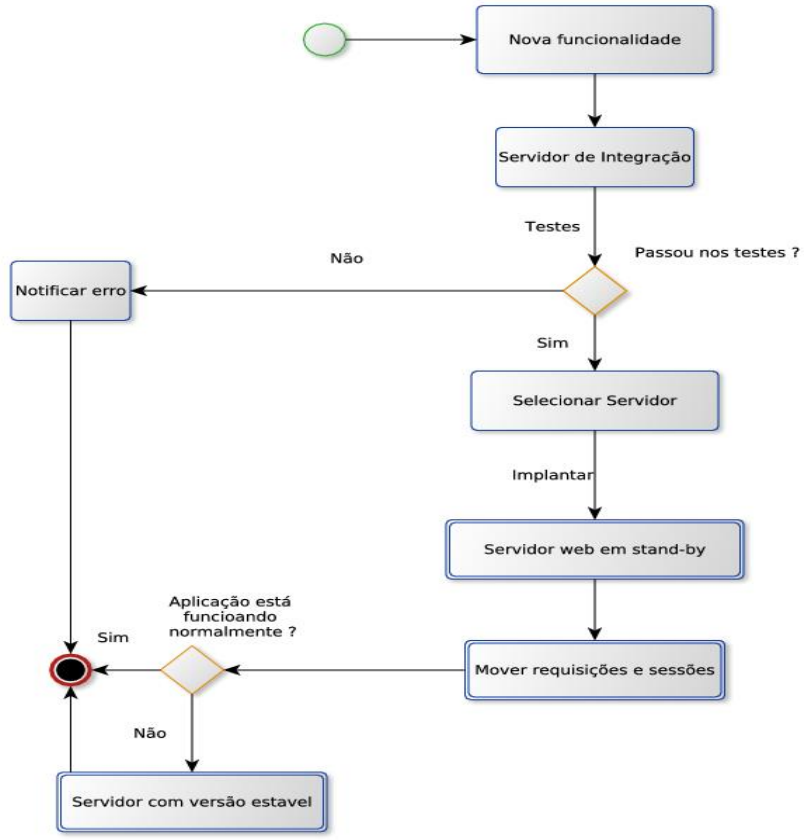

Figura 3 - Diagrama de atividade

A abordagem de virtualização descrita por [3] traz resultados concretos e sua descrição se encaixa com a estratégia *blue-green*, seus atrativos estão na forma dos resultados alcançados.

O ambiente virtualizado descrito por [3] é configurado por três ambientes:

O primeiro recurso virtualizado o balanceador de carga incumbido de detectar problemas de desempenho e erros que apenas seriam percebidos num ambiente de produção com acesso completo dos usuários, o qual não pode ser detectado em testes ou no ambiente de homologação. Esse componente é responsável por mover os clientes de um ambiente ao outro nas situações descrita acima, porém para atender de maneira mais efetiva a entrega contínua também deverá possuir mecanismo o qual ao ser realizado uma nova implantação moveria os clientes de forma automática para nova versão e não apenas em momentos de falhas.

O segundo e terceiro recurso virtualizado seria o próprio ambiente de produção e podem fazer o papel de ambientes *bluee* e *green*, esse componente é onde está instalado o servidor de aplicação. Por se tratar de ambiente virtualizado o servidor de qualquer fabricante poderá ser adotado.

Essa abordagem traz uma série de benefícios como:

Os recursos virtualizados poderiam ser descritos em forma de *script* para que fossem executados por processos automatizados, como sugerido por [1], e assim entrassem para um sistema de controle de versão.

Sendo o ambiente resultado de virtualização não haverá necessidade de aquisição de novos *hardwares* diminuindo assim o TCO.

Com o ambiente instrumentalizados como descrito por [3], assim que o *software* deixasse de atender os requisitos de desempenhos fixados ou falhas detectadas nos *logs*, os clientes serão movidos para versão antiga e estável.

Este abordagem está ancorado pelos resultados dos trabalhos dos autores [3] [8][9] onde demonstram um ambiente virtualizado que fornece um alto grau de disponibilidade, que apesar de se tratar sobre a questão do *software aging* pode ser aplicada à entrega contínua e adaptada a estratégia *blue-green*, na Tabela 4 demonstra os critérios para a seleção dessa abordagem.

Todos esses resultados alcançados estão alinhados a mnimização dos riscos de indisponibilidade e podem ser adaptados a estratégia *blue-green*.

Tabela 4 - Comparação de trabalhos correlatos na utilização de virtualização

Critério	[3]	[8]	[9]
Autorrecuperação	●	●	●
Aplicação em qualquer servidor	●		
Diminuição MTTR	●		
Virtualização	●	●	●
Sem perda de requisições ou sessões	●	●	
Executável em apenas um computador	●	●	●

O último ponto a se destacar, é que as abordagens combinadas descritas possuem pontos de aderência entre elas e o *framework* ITIL descantando-se a parte de transições de serviços, os itens de aderência são visto na Tabela 4:

Processos	*Blue-Green/Canary* & Virtualização/Virtualização Leve
Planejamento de transição e suporte	
Gerenciamento de mudança	
Gerenciamento de configuração e ativos de serviço	●
Gerenciamento de implantação e liberação	●
Teste e validação de serviços	●

Avaliação	●

A aderência entre o *framework* e as abordagens são descritas nas seguintes formas:

Gerenciamento de configuração e ativos de serviço: como já explicado por [1], tratando-se de ambientes virtualizados, todas as configurações e especificações dos ambientes utilizados podem ser descritos na forma de scripts e armazenados em sistemas de controle versão, assim toda e qualquer modificação será registrada podendo ser desfeita além de existir o histórico.

Gerenciamento de implantação e liberação: por se tratar de ambientes virtualizados as tarefas de implantação e liberação podem ser realizados através serviços automatizados. Cabendo apenas estabelecer tipo de permissões para controlar quem pode ou não executar a implantação e liberação.

Teste e validação de serviços: todas as etapas de teste e validação já estão devidamente estabelecida na entrega contínua descrita por [1] em seu pipeline, antes de chegar no processo de implantação e liberação propriamente, uma bateira de testes são executados e apenas quando executados e passados em todos com sucesso o serviço é disponibilizado para as etapas posteriores.

Avaliação: após a implantação no ambiente selecionado, poderão ser feitos testes de estresse a fim de verificar se atende os requisitos pré-definidos somente com a passagem nestes testes, é feito então a comutação entre os ambientes virtualizados. Essa etapa de comutação em casos mais específicos, podem ser realizados de forma manual apenas depois da aprovação do responsável pela liberação.

Processos não aderentes as abordagem:

Planejamento de transição e suporte e Gerenciamento de mudança: não se estão nas abordagem pois fogem do escopo das mesmas, pois ambos os processos têm foco em atividades anteriores a efetiva implantação de um novo serviço.

Essa abordagem seria a princípio para organizações que já possuem seus próprios equipamentos e poucos recursos iniciais para investimento, mas não só limitada a elas.

4.1 LIMITAÇÕES

Existem algumas limitações e desvantagens associada à proposta e cabe a devida atenção a elas.

A primeira é que apesar de não necessitar de novos recursos físicos, exige-se que a organização que deseja implementá-la precisará de pessoas com fortes conhecimentos em virtualização, para que funcione de forma adequada e esperada.

A segunda é que os clientes que usufruírem de novas funcionalidades e no meio de sua operação for movido para versão antiga por algum problema, deveram ser atingidos por uma indisponibilidade, pois a versão antiga não possuirá a funcionalidade usada e, portanto a sessão ou requisição não poderá ser reestabelecida, dessa forma eles perderão o trabalho em progresso.

A terceira é que ocorrência de falha nesse único servidor com os ambientes virtualizados acarretará a indisponibilidade do serviço de qualquer forma.

A quarta é as questões envolvendo o banco de dados e que não estão nas abordagens apresentadas, assim existe toda uma problemática envolvida que precisa ser resolvida em torno desse tema.

5 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho tentou fornecer um comparativo entre as possíveis soluções em relação ao problema de indisponibilidade, também foi feita uma sugestão mais concreta de como abordar a questão do *downtime*, e acredito que a técnica descrita pode ser um caminho a ser seguido. Essas abordagens apontadas de forma geral conseguem atender aquelas organizações que possuem poucos recursos, mas têm consigo a preocupação de melhor se colocar em relação a seus clientes. Por fim cabe salientar que o trabalho é teórico e precisa em um futuro ser implementado para fornecer resultados mais concretos aos possíveis utilizadores.

REFERÊNCIAS

[1] HUMBLE, J.; FARLEY, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. [S.l.]: Pearson Education, 2010. (Addison-Wesley Signature Series (Fowler)). ISBN 9780321670229.

[2] ZERO Downtime Database Upgrade Using Oracle GoldenGate. 2015. (Accessed on 06/28/2016). Disponível em:

<http://www.oracle.com/technetwork/middleware/goldengate/overview/ggzerodowntimedatabaseupgrades-174928.pdf>.

[3] SILVA, L. et al. Using Virtualization to Improve Software Rejuvenation. v. 3, n. 1, 2007.

[4] AKERELE, O.; RAMACHANDRAN, M.; DIXON, M. System dynamics modeling of agile continuous delivery process. In: Proceedings - AGILE 2013. [S.l.: s.n.], 2013. ISBN 9780769550763.

[5] NEELY, S.; STOLT, S. Continuous delivery? Easy! Just change everything (well, maybe it is not that easy). In: Proceedings - AGILE 2013. [S.l.: s.n.], 2013. ISBN 9780769550763.

[6] CHEN, L. Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software, v. 32, n. 2, p. 50–54, mar 2015. ISSN 0740-7459. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7006384>.

[7] BELLOMO, S. et al. Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail. In: Proceedings of the International Conference on Dependable Systems and Networks. [S.l.: s.n.], 2014. ISBN 9781479922338.

[8] THEIN, T.; CHI, S.-d.; PARK, J. S. Improving Fault Tolerance by Virtualization and Software Rejuvenation. p. 855–860, 2008.

[9] SHETTY, H.; NAMBIAR, M.; KALITA, H. Analysis and Application of Conditional Software Rejuvenation – A New Approach. 2008.

[10] GAVRILOVSKA, A.; SCHWAN, K.; OLESON, V. A Practical Approach for ‘Zero’ Downtime in an Operational Information System. 2002.

[11] FERNANDES, A.; ABREU, V. de. Implantando a Governança de TI – 4 a Ed.: Da estratégia à Gestão de Processos e Serviços. [S.l.]: Brasport, 2014. 495 p. ISBN 9788574526584.

[12] MOURA, B. Logística: Conceitos e Tendências. [S.l.]: CENTRO ATLANTICO, 2006. 148–149 p. ISBN 9789896150198.

[13] NETO, M. de S. Virtualização – 2ª Edição: Tecnologia Central do Datacenter. [S.l.: s.n.], 2015. 21–22 p. ISBN 9788574527611.

[14] AGARWAL, B.; TAYAL, S.; GUPTA, M. Software Engineering and Testing. [S.l.]: Jones & Bartlett Learning, 2010. 110 p. (Computer science series). ISBN 9781934015551.

[15] STAMOU, K.; KANTERE, V.; MORIN, J.-h. SLA data management criteria. p. 34–42, 2013.

[16] SHU, Z.; MEINA, S. An Architecture Design of Life Cycle Based SLA Management. p. 1351–1355, 2010.

[17] ESMAILI, H. B.; GARDESH, H. Strategic Alignment: ITIL Perspective. n. Icctd, p. 550–555, 2010.

[18] APPLICATIONS, E. S.; EIKEBROKK, T. R. ITIL implementation: The role of ITIL software and project quality. n. January, 2012.

[19] SERVICE transition. Stationery Office, 2007. (IT infrastructure library). ISBN 9780113310487. Disponível em: <https://books.google.com.br/books?id=a57syGFfEWkC>.

[20] FOWLER, M. Blue-Green Deployment. 2010. <http://martinfowler.com/bliki/BlueGreenDeployment.html>. (Accessed on 06/28/2016).

[21] SATO, D. Canary Release. 2014. <http://martinfowler.com/bliki/CanaryRelease.html>. (Accessed on 06/28/2016).

[22] LINUX Containers: Parallels, LXC, OpenVZ, Docker and More | au courant technology. <https://aucouranton.com/2014/06/13/linux-containers-parallels-lxc-openvz-docker-and-more/>. (Accessed on 07/11/2016).

[23] MORABITO, R.; KOMU, M. Hypervisors vs. Lightweight Virtualization: a Performance Comparison. p. 386–393, 2015.

[24] SAGANA, C. Performance Enhancement in Live Migration for Cloud Computing Environments. p. 361–366, 2013.

[25] VAUGHAN-NICHOLS, S. J. New Approach to Virtualization Is a Lightweight. p. 12–14.