Reginaldo Coimbra Vieira

Introduzindo Gerenciamento Autonômico de Incidentes em Sistemas de Software Legados

São João del-Rei

2021

Reginaldo Coimbra Vieira

Introduzindo Gerenciamento Autonômico de Incidentes em Sistemas de Software Legados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São João del-Rei – UFSJ, como requisito parcial para a obtenção do título de Mestre.

Linha de pesquisa: Sistemas de Informação

Universidade Federal de São João del-Rei – UFSJ

Ciência da Computação

Programa de Pós-Graduação

Orientador: Elder José Reioli Cirilo

São João del-Rei

2021

Reginaldo Coimbra Vieira
Introduzindo Gerenciamento Autonômico de Incidentes em Sistemas de Software Legados/ Reginaldo Coimbra Vieira. – São João del-Rei, 2021-
71p. : il. (algumas color.) ; 30 cm.

Orientador: Elder José Reioli Cirilo

Dissertação (Mestrado) – Universidade Federal de São João del-Rei – UFSJ
Ciência da Computação
Programa de Pós-Graduação, 2021.

1. Computação Autonômica. 2. Manutenção de Software. 3. Sistemas Legados. 4. Software Autoadaptativo. I. Cirilo, E. II. Universidade Federal de São João del-Rei. III. Ciência da Computação. IV. Título

Reginaldo Coimbra Vieira

Introduzindo Gerenciamento Autonômico de Incidentes em Sistemas de Software Legados

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São João del-Rei – UFSJ, como requisito parcial para a obtenção do título de Mestre.

Linha de pesquisa: Sistemas de Informação

Elder José Reioli Cirilo
Orientador

Marx Leles Viana
Pontifícia Universidade Católica do Rio de Janeiro

Daniel Ludovico Guidoni
Universidade Federal de São João del-rei

São João del-Rei
2021

Resumo

O projeto e desenvolvimento de sistemas de *software* autônomos vêm sendo amplamente investigados principalmente a partir do manifesto disseminado pela IBM sobre a Computação Autonômica (*Autonomic Computing*). Sistemas de *software* autonômicos são capazes de: adaptar os comportamentos atuais à medida que surgem outras demandas por novos requisitos; e de resolver incidentes com a mínima intervenção humana. Porém, investigações de como incorporar as capacidades de adaptação e resolução autônoma de incidentes em sistemas existentes (legados), apesar de desejado, tiveram pouco progresso nos últimos anos. Permitir que sistemas de *software* legados se adaptem requer a identificação e incorporação de atuadores responsáveis pela adaptação do comportamento e dos sensores que medem a variação no estado dos comportamentos monitorados. Idealmente, a incorporação de atuadores e sensores deve ser realizada de um modo flexível, isto é, sem que o sistema de *software* tenha que ser projetado e implementado novamente. Com essa finalidade, propomos uma nova abordagem para o gerenciamento autônomo de incidentes em sistemas de *software* legados. Nossa abordagem é baseada no conceito primordial de adaptação do comportamento por meio de conteinerização de sistemas. Com isso, é possível incorporar em sistemas legados novos comportamentos autônomos de modo rápido e flexível. Apresentamos adicionalmente uma avaliação e discussão da aplicabilidade da nossa abordagem no Moodle, um sistema legado largamente utilizado na prática para a condução de cursos a distância.

Palavras-chave: Computação Autonômica. Manutenção de Software. Sistemas Legados. Auto gerenciamento.

Abstract

Research about design and development of autonomous software systems has been widely done since the manifesto concerning Autonomic Computing disseminated by IBM. Autonomous (or autonomic) software systems are capable of: adapting current behaviors as other demand for new requirements arises; and solving incidents with minimal human intervention. However, research about how to incorporate the capabilities of adaptation and autonomous resolution of incidents into existing systems (legacy), although desired, have had little progress in recent years. Allowing legacy software systems to adapt themselves requires identification and integration of: actuators responsible for adapting the behavior; and sensors that measure the monitored resources status variation. Theoretically, actuators and sensors integration should be done in a flexible way, that is, without the software system having to be designed and implemented again. To this end, we propose a new autonomous incident management approach for legacy software systems. Our approach is based on the fundamental concept of behavior adaptation by using systems containerization. Therewith, it is possible to integrate new autonomous behaviors into legacy systems in a fast and flexible way. Moreover, we show an avaliation and a discussion about the deployment of our approach with Moodle, a legacy system widely used in practice for hosting online courses.

Keywords: Autonomic Computing. Software Maintenance. Legacy Systems. Self management.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AVA	Ambiente Virtual de Aprendizagem
DSL	Domain-specific Language (Linguagem Específica de Domínio)
ITIL	Information Technology Infrastructure Library (Biblioteca sobre Infraestrutura de Tecnologia da Informação)
JSON	JavaScript Object Notation (Notação de Objeto em *JavaScript*)
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
MIB	Management Information Base (Base de Informações Administrativas)
NMS	Network Management Station (Estação de Gerenciamento de Rede)
SDDC	Software-Defined Data Center (*Data Center* Definido por *Software*)
SDN	Software-Defined Network (Rede Definida por *Software*)
SDS	Software-Defined System (Sistema Definido por *Software*)
SDSec	Software-Defined Security (Segurança Definida por *Software*)
SGBD	Sistema de Gerenciamento de Banco de Dados
SNMP	Simple Network Management Protocol (Protocolo Simples para Gerenciamento de Rede)
SO	Sistema Operacional
VM	Virtual Machine (Máquina Virtual)
XML	Extensible Markup Language (Linguagem Extensível para Marcação)

Sumário

1 Introdução

A adoção acelerada de sistemas de *software* impulsionou o aumento da complexidade do que vêm sendo desenvolvido e a demanda por manutenções constantes (ZHU; PHAM, 2018). Isso exige que organizações implantem modelos operacionais com equipes autônomas e totalmente capacitadas para que cada sistema de software possua sua própria dinâmica de operação, manutenção e gerência. Conforme apontado por Martin Fowler (FOWLER, 2015), a dinâmica do mercado exige equipes operacionais cada vez mais maduras e aptas a gerenciar volumes consideráveis de serviços que são reimplantados regularmente.

Segundo a pesquisa *Allianz Risk Barometer 2020* conduzida pela seguradora alemã Allianz no final de 2019 e publicada no início de 2020 (ALLIANZ, 2020), incidentes cibernéticos constituem a principal preocupação para os negócios no mundo. Essa representa uma grande variação, porque, há sete anos, esse tipo de risco ocupou o 15º lugar, com apenas 6% das respostas. A referida pesquisa contou com 2.718 participantes de 102 países. Entre eles estavam clientes, consultores, especialistas e gerentes sêniores do segmento de seguro corporativo. Já a pesquisa conduzida pela Allianz, respondida por 83 participantes, em relação aos riscos para os negócios no Brasil, revelou que a preocupação com incidentes cibernéticos aparece em segundo lugar.

Apesar da preocupação com incidentes cibernéticos já se destacar em 1º lugar na lista, o seu nível percentual poderá revelar-se ainda superior na próxima pesquisa diante do surto pandêmico da Covid-19 desde o final de 2019. A demanda elevada e repentina para garantir o distanciamento social visando minimizar a propagação do vírus Sars-CoV-2 com o aumento da adoção do teletrabalho e da educação a distância faz a necessidade de garantir a disponibilidade e a segurança nos serviços de Tecnologia da Informação crescer. Com ela, riscos relacionados a problemas sanitários, que na pesquisa atual figurou na 17ª posição, também sinaliza uma iminente e acentuada progressão.

Aliados à complexidade, que é inerente aos sistemas de *software* modernos, interrupções do negócio e incidentes cibernéticos (e.g., interrupções, falhas graves) vêm se tornando cada vez mais onerosos às organizações, e, em contra partida, os usuários estão cada vez menos tolerantes a longos períodos de inatividade. Tipicamente, tudo isso exerce uma grande pressão nas equipes operacionais. Com isso, para conter a complexidade e seus efeitos, organizações precisam implementar estruturas operacionais que atuem sustentavelmente.

Se o serviço ficou indisponível por 60 minutos e foi corrigido em 5 minutos, o que aconteceu durante os outros 55 minutos? Isso pode ser decorrência de fraca automação (alertas, recuperação automatizada etc.), ferramentas de diagnóstico fracas, procedimentos

obscuros de escalonamento no suporte (atraso para conduzir ao grupo ou fornecedor de suporte técnico apropriado) ou falta de documentação operacional compreensível (HUNNEBECK, 2011). Este cenário leva a um enorme interesse no que alguns chamam de computação autonômica: em particular, a noção de sistemas de *software* autogerenciáveis é uma abordagem atraente para reduzir o tempo e os custos de operação e manutenção deles e aumentar seus níveis de confiabilidade e garantia. Atualmente, existem técnicas e tecnologias que visam automatizar a detecção de problemas e/ou a correção de problemas em sistemas de *software* (MONPERRUS, 2018). Por exemplo, sistemas operacionais incluem mecanismos para automatizar a coleta de dados sobre falhas assim como ferramentas de terceiros auxiliam na detecção de comportamentos anômalos pelo monitoramento de *logs* do sistema. No entanto, essas técnicas e tecnologias deixam a análise para um administrador humano sobre o quê, como e por quê o sistema está ou não fazendo algo. É o administrador quem deve determinar, planejar e executar a reconfiguração ou reparo. Por outro lado, sistemas de *software* autonômicos são capazes de adaptar os comportamentos existentes à demanda de novos requisitos e à resolução de incidentes com a mínima intervenção humana.

Entretanto, as abordagens descritas na literatura para o desenvolvimento de sistemas de *software* autonômicos, no geral, ignoram a existência do *software* legado (LI; CHEN; HASSAN, 2018). Podemos considerar que a migração geralmente não é uma opção, porque o futuro do negócio pode ficar em risco diante da crucialidade que eles desempenham para as organizações (BENNETT, 1995). Neste caso, ele pode necessitar perdurar. Mesmo que tenha sido implementado em uma linguagem de programação não mais em uso comum, que não há muitos mantenedores especialistas, isso aumenta ainda mais a necessidade de reparo autônomo. O termo “Sistemas Legados” está geralmente associado aos antigos sistemas monolíticos escritos em linguagens de programação obsoletas como FORTRAN e COBOL. Porém, mesmo sistemas de *software* modernos que não mais satisfazem a novos requisitos ou a demandas de qualidade são considerados igualmente sistemas de *software* legados. Esses sistemas representam anos de experiência e conhecimento acumulados, e muitos deles desempenham papéis cruciais em organizações (SCHNEIDEWIND; EBERT, 1998).

Porém, adicionar recursos de computação autonômica a esses sistemas existentes de larga escala é uma tarefa de engenharia de *software* particularmente desafiadora. Primeiro, uma vez que os sistemas foram projetados e construídos sem uma perspectiva de computação autônoma, a sua adoção nos sistemas existentes requer uma considerável investigação, e alterações e refatoração do código existente. Segundo, não é provável que as empresas corram o risco de reprojetar completamente o sistema ou executar alterações significativas no código, pois elas podem levar a consequências inesperadas (por exemplo, introduzir novos *bugs* funcionais ou não funcionais). Por isso, muitos sistemas legados são mantidos, principalmente, por serem estáveis e não apresentarem necessidades de mudanças

impactantes. Sua substituição resulta em investimento de tempo e dinheiro na migração, nas fases de testes e no treinamento. Este é o caso do Moodle, um Ambiente Virtual de Aprendizagem (AVA)[1]. Devido às suas 4.120.919 linhas de código[2] e aos seus 1.649 *plug-ins* disponíveis[3], reescrevê-lo mostra-se ineficiente e caro. Além disso, essa iniciativa ainda se mostra improvável considerando que uma comunidade de desenvolvedores necessita reescrever *plug-ins* e entender sobre os conceitos de computação autonômica a fim de preparar todo o sistema para se readaptar, por exemplo, após uma sobrecarga de requisições de usuários.

1.1 Objetivo Geral e Contribuições

O projeto e desenvolvimento de sistemas de *software* autônomos vêm sendo amplamente investigados principalmente a partir do manifesto disseminado pela IBM sobre a Computação Autonômica (*Autonomic Computing*). Sistemas de *software* autonômicos são aqueles capazes de adaptar os comportamentos existentes à medida que surgem novos requisitos e de resolver incidentes com a mínima intervenção humana. Porém, investigações de como incorporar as capacidades de adaptação e resolução autônoma de incidentes em sistemas existentes (legados), apesar de desejado, tiveram pouco progresso nos últimos anos.

Inicialmente, permitir que sistemas de *software* legados se adaptem requer a identificação e a incorporação de atuadores responsáveis pela adaptação do comportamento, e dos sensores que medem a variação no estado dos recursos monitorados. Por outro lado, a incorporação de atuadores e sensores deve ser idealmente realizada sem que o sistema de *software* tenha que ser projetado e implementado novamente.

O objetivo geral deste trabalho é o desenvolvimento de uma abordagem para incorporação de capacidades autonômicas para gerência de incidentes em sistemas de *software* legados sem alterar o código original. A abordagem proposta é baseada no conceito primordial de adaptação do comportamento por meio de conteinerização de sistemas. Com isso, é possível incorporar em sistemas legados novos comportamentos autônomos de modo rápido e flexível. Este trabalho apresenta as seguintes contribuições:

1. Uma abordagem para incorporação de capacidades autonômicas para gerência de incidentes em sistemas de *software* legados.

[1] AVA contém recursos essenciais para ministrar aulas a distância tais como provas, questionários, fóruns de discussões, tarefas e outras ferramentas que promovem a interação tanto entre os alunos quanto entre os alunos e o professor. Enquanto WordPress, Joomla e Drupal, por exemplo, focam na disponibilização do conteúdo para um público geral (são Sistemas de Gerenciamento de Conteúdo), AVAs focam na disponibilização de ambientes específicos para promover a aprendizagem.

[2] <https://www.openhub.net/p/moodle>

[3] <https://moodle.org/plugins/>

2. O *framework* Adapta, que implementa a abordagem para incorporação flexível de capacidades autonômicas para gerência de incidentes em sistemas de *software* legados.
3. Um estudo qualitativo sobre gerência de incidentes na prática.
4. Uma avaliação e discussão da aplicabilidade da abordagem no Moodle, um sistema legado de larga escala amplamente utilizado na prática para a condução de cursos *on-line*.

1.2 Organização

O conteúdo deste trabalho foi organizado nos seguintes capítulos:

- Capítulo 2: explica sobre alguns conceitos fundamentais para a compreensão deste trabalho;
- Capítulo 3: apresenta o ambiente real da Universidade Federal de São João del-Rei no qual ocorrem incidentes que afetam a disponibilidade do Ambiente Virtual de Aprendizagem (AVA) Moodle e realiza um estudo qualitativo em outras instituições públicas que também ofertam educação a distância;
- Capítulo 4: detalha sobre o funcionamento do *framework* Adapta;
- Capítulo 5: testa e apresenta o resultado sobre o uso do *framework* Adapta;
- Capítulo 6: apresenta outros recursos com os quais o *framework* Adapta também pode se integrar futuramente; e
- Capítulo 7: comenta sobre as principais limitações identificadas ao decorrer do desenvolvimento deste trabalho e quais desafios deverão ser superados para agregar ainda mais valor a esta abordagem.

2 Background

A seguir, encontram-se algumas definições fundamentais para compreender o objetivo de gerir incidentes em sistemas de *software* legados usando técnicas de conteinerização e computação autonômica.

2.1 Sistemas de *Software* Legados

O termo "Sistemas Legados" está geralmente associado aos antigos sistemas monolíticos escritos em linguagens de programação obsoletas como FORTRAN e COBOL (BENNETT, 1995). Porém, sistemas de *software* modernos (e.g., baseados nas últimas tecnologias Web) que não mais satisfazem a novos requisitos ou a demandas de qualidade são considerados igualmente sistemas de *software* legados. Esses sistemas representam anos de experiência e conhecimento acumulados, e muitos deles desempenham papéis cruciais em organizações (SCHNEIDEWIND; EBERT, 1998).

Consequentemente, conforme as organizações modernizam suas estratégias de negócio, torna-se necessário a evolução de seus sistemas de *software* também. Eles precisam amadurecer de acordo com novas demandas. Porém, qualquer evolução, mesmo que planejada cuidadosamente, pode comprometer a condução dos negócios organizacionais. Diante dos riscos associados à construção de sistemas de *software* novos e mais adaptados às realidades momentâneas (e.g., dinâmica de mercado, leis e demandas dos usuários), organizações optam por manter o legado atendendo às mínimas condições de uso. Isto é, quando o risco de reprojetar um sistema parecer muito elevado, preserva-se o legado (SCHNEIDEWIND; EBERT, 1998). Em geral, a migração não é uma opção viável. Então, o legado pode e vem perdurando mesmo que ele tenha sido implementado em linguagens obsoletas ou inseguras.

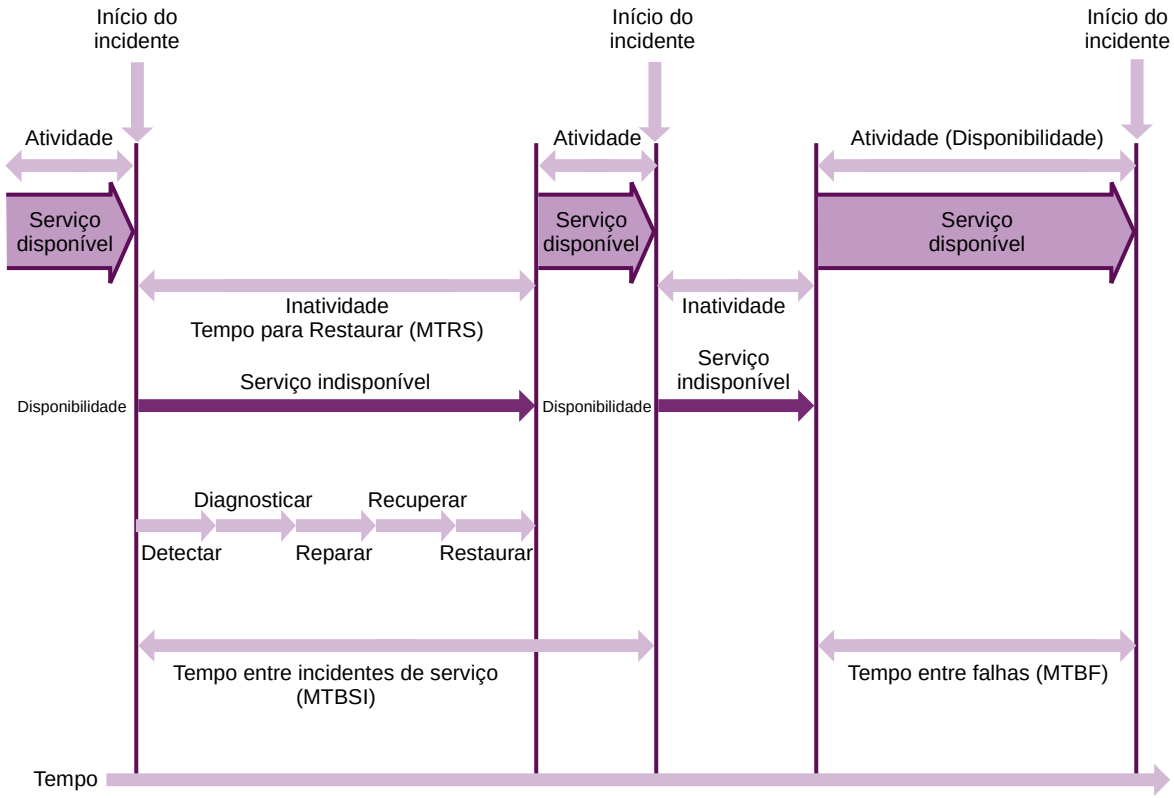
2.2 Gerenciamento de Incidentes

Uma interrupção não planejada em um serviço de Tecnologia da Informação (TI) ou a redução na sua qualidade é considerada como um incidente (ITSMF, 2012). Um incidente pode ser definido pela interrupção não planejada de uma atividade da organização, causando impactos negativos, como a perda da confiança dos clientes, a perda financeira e a queda na produtividade (ALLIANZ, 2020). O gerenciamento de incidentes é uma disciplina que visa monitorar e resolver interrupções de serviço com rapidez e eficiência, permitindo que as equipes operacionais se concentrem no que agrega valor para a organização ao invés de se dedicarem a monitorar e a resolver eventos isolados. Ou seja, o objetivo do

gerenciamento de incidentes é restaurar serviços minimizando o impacto nas atividades comerciais e evitando perdas vultosas.

Gerência de incidentes cobre todos esses tópicos no auxílio à gerência da complexidade em sistemas de *software* criando um ambiente saudável para que operadores possam lidar com eles (ITSMF, 2012). Esforços manuais são um dos principais custos de operação e manutenção de modernos sistemas de *software* em larga escala.

Figura 1 – Ciclo de vida do incidente no processo de Gerenciamento da Disponibilidade da ITIL



A Figura 1 ilustra o ciclo de vida do incidente no processo Gerenciamento da Disponibilidade da ITIL (HUNNEBECK, 2011). O cliente continua satisfeito com o serviço mesmo quando incidentes acontecem, porém é necessário garantir que a sua duração seja minimizada (HUNNEBECK, 2011). Todo incidente passa por vários estágios e o tempo dispensado em cada um pode variar consideravelmente:

- *Detecção:* momento que a organização se torna ciente da ocorrência de um incidente. Ferramentas de gerenciamento de sistemas influenciam positivamente na habilidade de detectar eventos e incidentes, e, consequentemente, melhoram os níveis de disponibilidade que são entregues. Idealmente, o evento deve ser automaticamente detectado e resolvido antes que os usuários o percebam ou sejam impactados;

- *Diagnóstico:* determina a causa subjacente. O uso e a capacidade de ferramentas e habilidades de diagnóstico são críticas para acelerar a resolução dos problemas. O nível de diagnóstico pode estender o tempo de inatividade da prestação do serviço para certos tipos de falhas. Porém, a não realização de captura apropriada cria e expõe o serviço a falhas recorrentes. Deve-se garantir que apenas os dados essenciais de diagnóstico sejam obtidos;
- *Reparo:* momento que a falha foi reparada. A duração para reparo de incidentes deve ser continuamente monitorada e comparada com os acordos firmados;
- *Recuperação:* momento que a recuperação do componente foi concluída. Sempre que possível, as atividades operacionais dentro do planejamento de recuperação devem ser automatizadas. A procura contínua e a promoção de métodos mais rápidos para recuperação de todos os incidentes possíveis pode ser alcançada por meio de uma variedade de métodos incluindo detecção automatizada de falhas, recuperação automatizada etc.; e
- *Restauração:* momento que o serviço normaliza e, consequentemente, o incidente pode ser considerado "fechado". É importante que o serviço de TI seja verificado assim que a restauração conclua.

Há métrica para medir tanto o tempo de atividade quanto de inatividade de um serviço. Idealmente, para garantir a satisfação de clientes em relação à disponibilidade, deve-se maximizar o Tempo Médio entre Falhas (MTBF) e, quando qualquer incidente ocorrer, objetivar atingir o menor Tempo Médio para Restaurar o Serviço (MTRS).

O tempo associado a cada estágio do ciclo de vida do incidente influencia o tempo total de inatividade percebido pelo usuário. Segundo a ITIL (HUNNEBECK, 2011), considerando esta abordagem, é possível identificar em qual deles o tempo está sendo "perdido". Por exemplo, se o serviço ficou indisponível por 60 minutos e foi corrigido em 5 minutos, o que aconteceu durante os outros 55 minutos? Isso pode ser decorrência de fraca automação (alertas, recuperação automatizada etc.), ferramentas de diagnóstico fracas, procedimentos obscuros de escalonamento no suporte (atraso para conduzir ao grupo ou fornecedor de suporte técnico apropriado) ou falta de documentação operacional compreensível.

2.3 Computação Autonômica

A complexidade no gerenciamento e operação dos sistemas de *software* se aproxima continuamente dos limites da capacidade humana. Há, na atualidade, uma demanda crescente por novos meios de configuração, ajuste e distribuição de sistemas de *software* (KEPHART; CHESS, 2003). O aumento da atenção necessária devido ao crescente

número de recursos envolvidos na operação de sistemas complexos e heterogêneos tais como máquinas virtuais, servidores Web, servidores de aplicação, banco de dados, infraestrutura de rede, sistema operacional etc, torna o gerenciamento cada vez mais ineficiente e oneroso. É em meio a esta realidade que emerge a computação autonômica (IBM, 2006) e o manifesto disseminado pela IBM nos anos 2000. Nela, os sistemas de *software* monitoram continuamente o seu próprio comportamento e, com base em políticas de adaptação, reagem a incidentes externos que comprometem os seus objetivos. O benefício é que profissionais e desenvolvedores podem manter o foco em tarefas de maior valor para a organização enquanto o próprio *software* controla o seu comportamento e reage em resposta a interrupções e a mudanças no ambiente.

Sistemas de *software* autonômicos incorporam controles que podem ser divididos em quatro áreas funcionais de auto-gerenciamento (IBM, 2006):

- *Auto-configuração:* adaptação dinâmica em resposta a variações no ambiente. Adaptações podem incluir a implantação ou a remoção de componentes, ou mudanças nas características do sistema;

- *Auto-cicatrização:* correção de falhas sem interromper o próprio sistema de *software*. Neste caso, o sistema torna-se resiliente a incidentes;

- *Auto-otimização:* ajuste automático dos recursos do sistema para garantir que o serviço atenda às expectativas de negócio e do usuário final. Neste caso, os ajustes também contemplam a incorporação de novos recursos, por exemplo, para assegurar que transações de negócio possam ser concluídas dentro do tempo; e

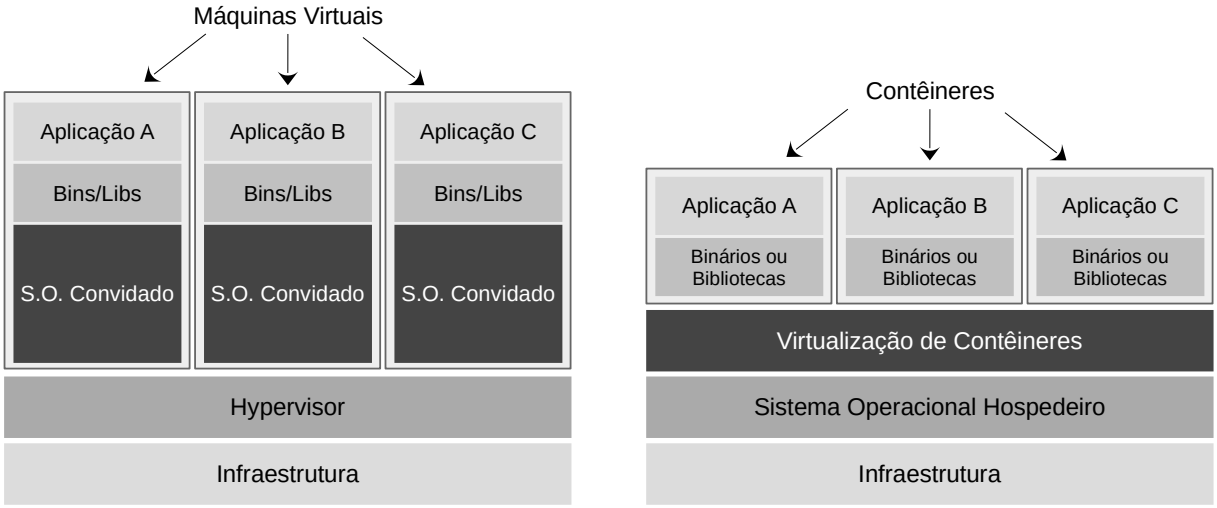
- *Auto-proteção:* ajuste das políticas de segurança e privacidade. Por exemplo, o sistema de *software* detecta comportamentos hostis à medida que eles ocorrem – podem incluir acesso não autorizado, proliferação de *malware* e ataques de negação de serviço – e os ameniza ou os evita para manter a integridade das informações corporativas.

Usualmente, os sistemas autonômicos podem tratar as áreas funcionais de autogerenciamento distintamente ou elas podem ser incorporadas como propriedades emergentes em uma arquitetura geral (KEPHART; CHESS, 2003). Para realizar os experimentos, o *framework* foi desenvolvido para abordar principalmente as propriedades de auto-cicatrização e auto-otimização, ou seja, o intuito foi resguardar que o sistema continuasse em operação após instabilidades no ambiente no qual o *software* opera e readaptar o ambiente para atender uma demanda crescente por acesso. Os dados desse ambiente são coletados pelos sensores e as modificações são realizadas pelos atuadores conforme as políticas de adaptação informadas pelo administrador.

2.4 Máquinas Virtuais e Conteinerização

Máquinas Virtuais (ou Virtual Machines – VMs) é um dos principais suportes à operacionalização de infraestruturas. Apesar do conceito de conteinerização ser semelhante ao da virtualização de máquinas, ela é uma solução mais leve e que exige menos recursos (DUA; RAJA; KAKADIA, 2014). Portanto, tornou-se a solução padrão para empacotamento de sistemas de software interoperáveis. Embora VMs e contêineres sejam técnicas de virtualização, elas resolvem problemas distintos. A técnica de conteinerização contém ferramentas próprias para o empacotamento de sistemas de *software* - ou seja, eles têm um foco em plataforma como serviço (PaaS) - de modo a promover maior interoperabilidade enquanto utiliza os princípios de virtualização do sistema operacional.

Figura 2 – Contraste entre Máquinas Virtuais e Conteinerização



A implantação baseada em VMs é ideal quando sistemas de *software* em uma mesma infraestrutura de implantação requerem sistemas operacionais distintos ou várias versões de um SO (e.g., Windows Server, Debian Linux, Ubuntu Linux e Unix). Neste caso, para fornecer essa capacidade de executar diferentes versões do sistema operacional é necessário operar em nível de VMs. Em conteinerização, os sistemas de *software* compartilham o mesmo sistema operacional e, como resultado, essas implantações terão um tamanho significativamente menor em relação às implantações em VMs, que necessitam ter um sistema operacional completo instalado em cada VM. Consequentemente, as técnicas de conteinerização possibilitam o armazenamento de centenas de contêineres em um servidor (Figura 2).

Uma imagem pronta para ser implantada em um contêiner contém sistemas de *software* empacotados, independentes e prontos apenas com os binários e bibliotecas necessárias. Essa tecnologia permite múltiplas instâncias no espaço de usuário serem executadas em um único *host* e é frequentemente chamada de virtualização em nível

de sistema operacional (TURNBULL, 2014). O ecossistema de contêineres consiste em um mecanismo de execução capaz de isolá-los do sistema operacional hospedeiro, e uns dos outros. Os repositórios de imagens desempenham um papel central no fornecimento de acesso a milhares de sistemas reutilizáveis. Além disso, o mecanismo de execução de contêiner provê uma API que permite criar, definir, compor e distribuir contêineres, executar ou iniciar imagens, e executar comandos nos contêineres. Também, há a possibilidade de distribuição de um único sistema de *software* em diversos contêineres usando a arquitetura de microsserviços.

Docker, por exemplo, permite que desenvolvedores empacotem sistemas de *software* com todas as dependências em imagens de modo que os mesmos poderão ser executados em qualquer outro computador, independentemente de qualquer configuração personalizada que possa diferir da máquina usada para escrever e testar a aplicação. É possível criar, iniciar, parar ou excluir contêineres usando a API do Docker. Além do modo de operação tradicional de conteinerização de sistemas de *software* em uma máquina, o Docker também pode trabalhar em *cluster* com outros nós da rede no modo Swarm. Nele, os nós são divididos em dois tipos: *managers* e *workers*. Um *cluster* não precisa ter nós *workers*, porque os gerentes, por padrão, também podem desempenhar o papel de execução. Entre os *managers*, há sempre o líder do *cluster*. Todos os comandos de gerenciamento executados em outros *managers* são automaticamente redirecionados para o líder a fim de que a operação desejada seja bem orquestrada entre todos os nós participantes.

3 Gerenciamento de Incidentes

Para ilustrar o contexto ao qual este trabalho se aplica, apresentamos e discutimos nesta seção casos reais e práticos de incidentes.

3.1 Incidentes – Um Contexto Prático

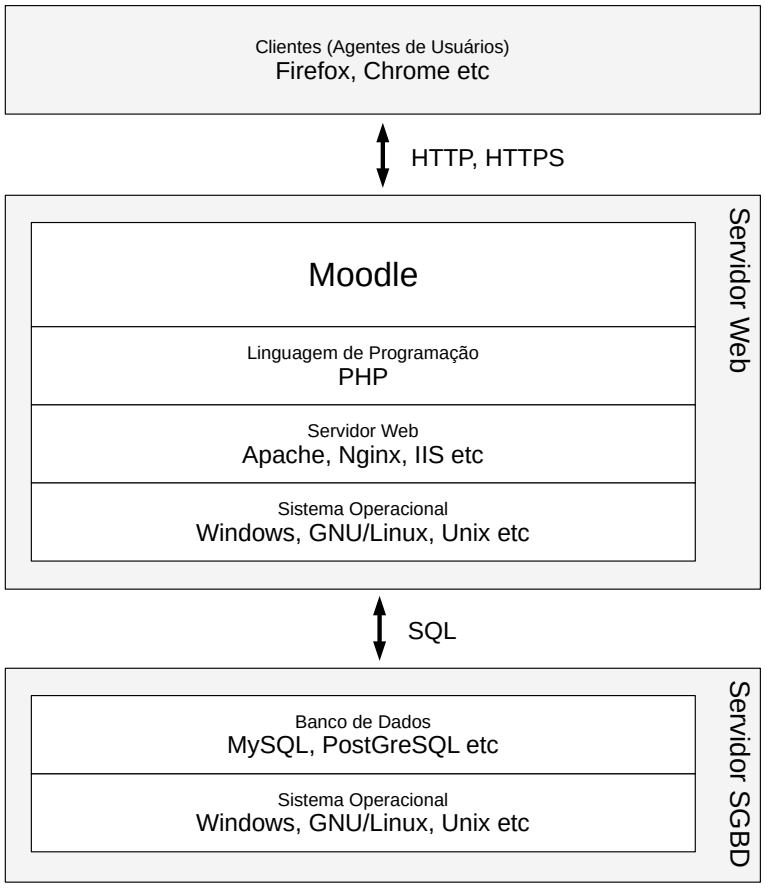
Nossa investigação se concentra em instâncias do Moodle[1] instaladas no Núcleo de Educação a Distância da Universidade Federal de São João del-Rei e que atendem aos cursos on-line. A Figura 3 mostra a arquitetura do Moodle. Um discente acessa o ambiente por meio de um agente Web (e.g., Mozilla Firefox, Google Chrome etc) via HTTP ou HTTPS. Desenvolvido na linguagem de programação PHP, o Moodle segue uma arquitetura distribuída (cliente-servidor) que opera sobre um servidor Web e persiste os dados em um Sistemas de Gerenciamento de Banco de Dados (SGBD). O exemplo (Figura 3) ilustra uma implantação onde o servidor Web está implantado em um ambiente que executa isoladamente daquele com o SGBD (e.g., em máquinas físicas, máquinas virtuais, contêineres Docker).

Semelhante a outros sistemas não-autonômicos de grande escala, o Moodle exige esforços manuais e caros para operação e manutenção. Mesmo com a crescente popularidade do DevOps (desenvolvimento e operações), esses esforços manuais, em certos casos, acabam não sendo tratados por desenvolvedores que têm um conhecimento profundo sobre o sistema, mas sim por aqueles contratados para implantação e operação do sistema. Então, embora a equipe de implantação e operação esteja familiarizada com o Moodle, eles ainda têm dificuldades em encontrar as configurações ideais relacionadas ao desempenho, por exemplo. Existem várias razões. Primeiro, como o Moodle não foi construído com os recursos de computação autonômica em mente, os valores de configuração geralmente são descritos estaticamente (por exemplo, em arquivos de configuração estáticos), o que aumenta o desafio do ajuste de configuração em tempo real. Segundo, questões de desempenho são sensíveis ao ambiente de implantação (por exemplo, largura de banda ou latência da rede) e à carga de trabalho (por exemplo, a quantidade de solicitações do usuário, o que muda constantemente e, em certos momentos críticos, apresenta altas variações). Portanto, operar o Moodle de modo que ele se comporte sempre dentro do desempenho ideal é extremamente caro e requer atenção constante da equipe de operação e implantação.

Os principais problemas que acontecem com o Moodle na Universidade Federal de São João del-Rei estão relacionados à lentidão no carregamento das páginas e a falhas

[1] Sistema *open-source*, personalizável, popular, estável, gratuito, baseado em *plug-ins* e que oferece um ambiente de aprendizagem. Pode ser obtido no endereço <http://moodle.org/>.

Figura 3 – Estrutura Genérica de Operação do Moodle



de comunicação entre os serviços necessários para atender às requisições de usuário (por exemplo: entre o SGBD e o Servidor Web, ou entre o Servidor Web e o PHP-FPM[2]). Embora elas sejam facilmente solucionáveis com pequenos ajustes e reinícios, a disponibilidade não pode ser comprometida por muito tempo devido à necessidade de intervenção humana. Normalmente, alunos, professores ou colaboradores do Núcleo de Educação a Distância identificam quaisquer problemas durante suas tentativas de acessos e os relatam ao suporte técnico, ou seja, o monitoramento também não é automatizado. Instabilidades também são ocasionadas por picos de acesso decorrentes de eventos organizados pelos professores no sistema (tais como provas *on-line*, definição de tarefas e uso de *plug-in* que consome muitos recursos) sem notificar previamente ao suporte técnico. E mesmo que notificassem, é impraticável prever a quantidade de acesso ou ajustar os serviços manualmente para atendê-los satisfatoriamente sem pré-alocar muitos recursos desnecessariamente ou alocá-los com escassez. Portanto, além de reparar serviços que param de responder, é imprescindível a identificação automatizada de picos momentâneos no tráfego a fim de adequar o ambiente

[2] Execução de códigos em PHP por meio de um processo independente ao invés de um módulo compilado e acoplado ao Servidor Web. É necessário haver essa independência dependendo do modo de operação do Servidor Web. Um exemplo prático seria configurar o Servidor Web Apache para atender às requisições usando *threads* (ou seja, usando o módulo "worker" ou "event" ao invés do padrão "prefork", que trata as requisições recebidas por meio de processos) para dar suporte ao protocolo HTTP/2.

para melhor atender às requisições dos usuários.

3.2 Estudo Qualitativo - Gerência de Incidentes na Prática

O objetivo geral deste trabalho consiste no desenvolvimento de um modelo para incorporação de computação autonômica em sistemas legados. Para melhor desenvolver o modelo, investigamos qualitativamente uma questão de pesquisa central: **RQ**. Como equipes de operacionalização caracterizam os desafios na gerência de incidentes? A Listagem 1 resume as metas do estudo usando um modelo GQM (Metas / Perguntas / Métricas) (WOHLIN et al., 2012).

Analisar gerenciamento de incidentes
com o propósito de caracterizar
com respeito a tipos de incidentes, responsáveis pela gerência, modelo de identificação e modelo de resolução
do ponto de vista de membros de equipes operacionais
no contexto do Moodle como AVA em Instituições de Ensino.

Os métodos qualitativos parecem adequados para responder a essa pergunta, porque analisa as relações entre conceitos e lidam diretamente com a complexidade existente. Como os resultados qualitativos são, portanto, muito ricos e informativos, eles permitem focar na lógica dos participantes. Como método concreto para melhor compreender como equipes operacionais caracterizam os desafios na gerência de incidentes, realizamos entrevistas semiestruturadas com 12 profissionais, membros de equipes operacionais em instituições de ensino. Escolhemos entrevistas semiestruturadas, porque elas fornecem uma agenda básica, mas também permitem uma adaptação dinâmica com base nas respostas, ou seja, são projetadas para extrair não apenas as informações previstas, mas também tipos inesperados de informações.

3.2.1 Metodologia

Preparação para a coleta de dados. Criamos um documento sobre a entrevista descrevendo o processo e os objetivos do questionário que foi enviado aos participantes para familiarizá-los com o estudo. Ele também abordou considerações éticas, como confidencialidade, solicitou o consentimento dos participantes e comunicou que as respostas seriam publicadas anonimamente. Para a condução do questionário, criamos um formulário *on-line* que continha as questões agrupadas em blocos. No Apêndice A, apresentamos o questionário utilizado na condução das entrevistas que inclui perguntas sobre a experiência das equipes de operacionalização na lida com incidentes e com questões sobre o perfil dos

Tabela 1 – Perfil de Experiência dos Participantes

Participante	Administração de Sistemas	Banco de Dados	Programação	Moodle ou outro AVA
P1	mais de 10 anos	mais de 10 anos	de 2 a 3 anos	mais de 10 anos
P2	mais de 10 anos	mais de 10 anos	mais de 10 anos	de 7 a 10 anos
P3	mais de 10 anos	sem experiência	sem experiência	mais de 10 anos
P4	mais de 10 anos	mais de 10 anos	mais de 10 anos	mais de 10 anos
P5	de 4 a 5 anos	de 4 a 5 anos	de 1 a 2 anos	de 4 a 5 anos
P6	de 7 a 10 anos	de 7 a 10 anos	de 7 a 10 anos	de 5 a 7 anos
P7	de 1 a 2 anos	de 1 a 2 anos	de 1 a 2 anos	mais de 10 anos
P8	de 5 a 7 anos	de 5 a 7 anos	de 5 a 7 anos	de 5 a 7 anos
P9	de 7 a 10 anos	de 7 a 10 anos	de 7 a 10 anos	de 5 a 10 anos
P10	mais de 10 anos	mais de 10 anos	mais de 10 anos	mais de 10 anos
P11	de 7 a 10 anos	de 3 a 4 anos	6 meses	sem experiência
P12	de 7 a 10 anos	de 7 a 10 anos	de 7 a 10 anos	de 7 a 10 anos

entrevistados. Como participantes da entrevista, queríamos profissionais de operacionalização com experiência prática significativa e sólido conhecimento de administração de sistemas distribuídos. Com isso, recrutamos os participantes via convite encaminhado por *e-mail* a Núcleos de Educação a Distancia. Para alcançar uma perspectiva heterogênea e aumentar a riqueza de informações nos resultados, consultamos profissionais alocados em diversas instituições. No total, enviamos 70 convites dos quais 12 profissionais de operacionalização aceitaram participar do estudo e responderam o questionário. A Tabela 1 apresenta as características dos participantes. Para manter o anonimato, referimos os participantes como P1 – P12 (primeira coluna).

Análise dos dados. O primeiro passo foi uma análise individual das respostas de cada participante. As observações relevantes de cada questão foram extraídas da resposta e documentadas. Com base neste primeiro levantamento, usamos a análise de casos cruzados para identificar importantes generalizações e resumos. Documentamos, por fim, as tendências que foram agregadas em conclusões para cada questão da pesquisa. Os artefatos e os resultados da entrevista estão disponíveis em nosso repositório *on-line*.

3.2.2 Resultados e Discussões

Tipos de Configuração. No geral, os participantes apontaram massivamente o uso de um modelo tradicional de implantação do Moodle de modo que quatro participantes responderam que utilizam máquinas diferentes para o Servidor Web e o Sistema de Gerenciamento de Banco de Dados, e sete, dos doze participantes, responderam que utilizam uma mesma máquina tanto para o Servidor Web quanto para o Sistema de Gerenciamento de Banco de Dados. Apenas um dos participantes apontou que utiliza uma abordagem moderna de computação em nuvem por meio do uso de VM no Google Cloud.

Tipos de Incidentes e Tempo de Correção. Na prática, os participantes apontaram que levam, em média, de 30 minutos a 2 horas para resolverem qualquer incidente observado em produção. Os incidentes comumente observados são: problema de conexão com o SGBD (devido à sobrecarga no SGBD); sobrecarga no servidor Web; vulnerabilidades no Ambiente Virtual de Aprendizagem (AVA) ou nos seus módulos; e problemas de conexão com a Internet.

Identificação dos Incidentes. Questionados sobre como ocorre a identificação da origem de um incidente, a maioria dos participantes respondeu que recorrem observações manuais conduzidas diretamente ou indiretamente pela equipe de operacionalização, conforme destacou o participante P1: *"Há inspeção diária de todas as instalações Moodle, porém 90% dos problemas são identificados pelos usuários que entram em contato com o suporte. Partimos então das mensagens de erro apresentadas pelo próprio Moodle para identificar o ocorrido, simulamos acessos dos usuários para reproduzir o problema e então após identificar a causa, pesquisamos soluções."*. Um dentre os entrevistados apontou o uso de plataformas avançadas que proveem serviços de diagnóstico (monitoramento, registro, rastreamento, relatório de erros e alerta). O participante P6 respondeu: *"Uso o Stackdriver e Zabbix para monitoramento, sou alertado automaticamente."*. Porém, mesmo aplicando recursos que auxiliam na detecção automática de incidentes, o entrevistado destacou que, no geral, a equipe leva de 30 a 60 minutos para resolver qualquer incidente relatado.

Responsável pela Solução do Incidente. No geral, as abordagens adotadas variam dentre os ambientes conforme o tamanho da empresa. As soluções dos incidentes podem estar concentradas em um único indivíduo, com ou sem experiência na plataforma conforme apontado pelo participante P3: *"Técnico de TI que não tem expediência com moodle."*. Ou, em certos cenários, uma estrutura mais completa se mostra implantada: *"Temos 3 perfis e níveis para atendar problemas no AVA Moodle: (i) Nível 1: analista de help desk com ótimos conhecimento de usabilidade do Moodle (mais de 3 anos de experiência); Resolve 85% dos incidentes sozinho.; (ii) Nível 2: analista de infraestrutura, suporte e redes com mais de 5 anos de experiência em instalação e configuração de servidores, redes e serviços; Resolve 10% dos incidentes com o nível 1; (iii) Nível 3: desenvolvedor de AVA Moodle, com mais de 5 anos de experiência em programação em PHP, Javascript, etc para o ambiente Moodle. Resolve 5% dos incidentes com o nível 1 e 2"*.

Tecnologias de Implantação e Empacotamento. O uso de contêineres ou demais tecnologias de implantação (e.g. NFS, SMB, FTP, volumes compartilhados entre vários contêineres do Docker) não se destacou em relação à implantação tradicional[3] do ambiente Moodle. Quatro, dos doze participantes, destacaram que utilizam máquinas virtuais. Por exemplo, o participante P2 descreveu: *"Utilizamos 3 máquinas virtuais para 3 instâncias do moodle. Utilizamos um servidor dedicado para videoconferências*

[3] Única máquina física que mantém tanto o Moodle quanto o SGBD.

(BigBlueButton) Utilizamos uma máquina virtual para repositório de videoaulas.”. Já o participante P1 afirmou: *“Todas as nossas máquinas são virtualizadas e utilizamos bastante o FTP para acesso aos conteúdos nas máquinas.”*

Componentes Suscetíveis a Incidentes. Questionados sobre quais módulos são mais suscetíveis a incidentes no ambiente de produção, os participantes, no geral, relataram problemas em uma série de módulos e *plug-ins*, principalmente aqueles associados a atividades com um prazo determinado para entrega, como aponta o participante P3: *“Questionário e outras atividades com prazo determinado.”*; e o participante P8: *“Sem dúvida os módulos mais problemáticos são os de envio de tarefas e avaliações online (questionários), que demandam ou tempo limitado para preenchimento ou uma conexão razoável de internet, o que muitas vezes não acontece nos polos de nossos alunos (são cidades carentes e com infraestrutura modesta). Outro grande problema do moodle é seu esquema de armazenamento de cache e backup, que com o tempo fica extremamente inflado, sendo impossível de lidar caso o Moodle tenha muitos usuários e seja muito antigo.”*. Os participantes também destacaram problemas com: (i) *plug-ins* desenvolvidos pela própria instituição, como o P2: *“Tivemos alguns problemas de performance com plugins e ferramentas desenvolvidas por nós para o Moodle, mas os mesmos foram corrigidos em menos de 3 dias pelo nível 3 - programação e correção de bugs.”*; e (ii) com *plug-ins* de terceiros não homologados, conforme destaca o participante P12: *“Especialmente módulos ainda não incorporados pela Comunidade Moodle.org.”*. Outros *plug-ins* mais específicos também foram destaques, como apontam os participantes P9: *“Videoconferência. A utilização de vídeos aulas alocadas direta no Ambiente e os trabalhos sobrecarrega o espaço em disco.”*; e P11: *“O conversor de documentos Unoconv. Este é um executável que é capaz de converter formatos de documentos suportados pelo LibreOffice. Esse plugin gera sobrecarga no sistema com consumo excessivo de memória RAM acarretando lentidão e sobrecarga no sistema.”*

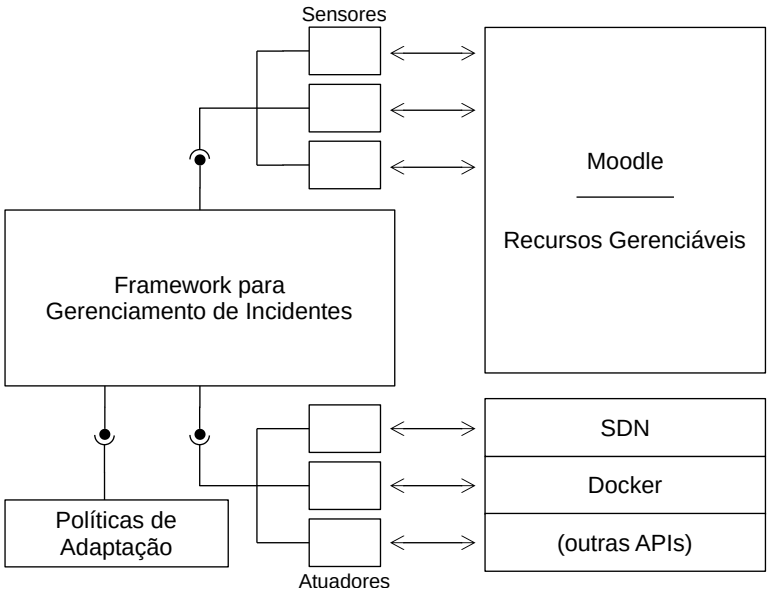
4 *Framework* Adapta - Gerenciamento Autonômico de Incidentes

Frameworks são responsáveis por fornecer o sistema e a arquitetura de *software* mais básicos. Esses projetos arquiteturais de *software* proporcionam confiabilidade, escalabilidade, extensibilidade e manutenibilidade, e os desenvolvedores confiam em *frameworks* específicos para implementar aplicações e lógicas de negócio mais complexas. Eles desenvolvem um conjunto de especificações, e os programadores trabalham sob elas.

Este capítulo explica como o *Framework Adapta* se organiza e cumpre seu objetivo principal: gerenciar incidentes. Sua arquitetura possui um conjunto básico de ferramentas para possibilitar o administrador de interagir com ele e para identificar o que acontece no ambiente monitorado, tomar decisões sobre como reagir e, consequentemente, aplicá-las.

Quando um site ou aplicativo implantado na computação em nuvem necessitar de poder computacional adicional, ele pode ser escalado. Os recursos alocados podem não ser suficientes para atender as demandas durante todo o tempo. Essa é uma consequência do sub-provisionamento que pode afetar os critérios da Qualidade de Serviço (QoS). Por outro lado, recursos alocados excessivamente para atender a demanda rotineira ocasiona custos desnecessários devido ao super-provisionamento (MIRESLAMI et al., 2019). Portanto, a elasticidade é interessante para que o provisionamento seja frequentemente otimizado de acordo com as demandas pelo serviço (COUTINHO et al., 2016).

Figura 4 – Interações no Framework



4.1 Arquitetura

A Figura 4 ilustra a arquitetura do *framework* Adapta. Informações sobre os recursos gerenciados (e.g., servidores Web, Moodle, módulos de *software* e banco de dados) são coletadas pelo *framework* por meio de sensores posicionados entre eles. Sensores expõem propriedades que representam o estado atual de um recurso gerenciável. Baseando-se nas políticas de adaptação definidas para um cenário específico, o *framework* Adapta pode readaptar a infraestrutura por meio dos atuadores para refletir as condições esperadas para o ambiente de execução do sistema de *software*. Então, os atuadores se responsabilizam pelo cumprimento dessas decisões. Eles têm a responsabilidade de interagir com componentes sobre os quais o software foi configurado para operar a fim de solucionar os incidentes identificados. Sucintamente, ele obtém informações sobre os recursos gerenciáveis usando sensores e modifica o comportamento dos componentes usados pelo *software* por meio dos atuadores.

O *framework* precisa ficar sempre em execução para obter informações pelos sensores, analisar os resultados e agir. As atividades aparentam, portanto, estar em um *loop* de controle[1] (HUEBSCHER; MCCANN, 2008) (RUTTEN; MARCHAND; SIMON, 2017). O ciclo começa com a coleta de informações sobre os serviços pelos sensores, prossegue com a tomada de decisão de acordo com as políticas aceitáveis, e realiza a aplicação delas por meio dos atuadores. Ele gerencia o *modus operandi* dos sensores e dos atuadores por meio das suas decisões baseadas nas políticas de adaptação informadas pelo administrador de sistemas. No caso deste trabalho, consideramos sistemas de *software* operando com base em instâncias de contêineres Docker, porém, na verdade, o *framework* tem modularidade suficiente para ser integrado com qualquer outro sistema. A Figura 4 ilustra atuadores integrados com Docker por meio da API fornecida.

Com base na infraestrutura de empacotamento de imagem e execução de contêiner oferecida pelo Docker, o *framework* Adapta reagirá ajustando o número de contêineres ao identificar um aumento no número de requisições visando prevenir o sistema de sofrer interrupções e instabilidades iminentes. Neste caso, o *framework*: obtém informações sobre o estado de operação dos contêineres incluindo, por exemplo, tempo de resposta, a quantidade de memória e processamento usados; toma uma decisão considerando as políticas de adaptação definidas; e atua sobre o *cluster* em Docker Swarm para aumentar ou reduzir o número de instâncias da aplicação.

[1] Conhecido como *loop* MAPE-K (Monitoração, Análise, Planejamento, Execução, Conhecimento).

4.2 Componentes e Implementação

4.2.1 Interface de Usuário para Administração do *Framework*

Para facilitar a sua administração remotamente pela Web por meio de uma interface de usuário amigável, ele foi integrado à linguagem PHP usando o recurso `FastCGI Process Manager` (PHP-FPM). Neste caso, após o *framework* ser carregado, ele aguarda por conexões de um agente que usa o protocolo HTTP, ou seja, qualquer navegador Web. Assim, é possível ajustar as configurações do *framework* usando uma interface simples.

Quando um sensor, atuador ou uma política de adaptação é instalada, modificada ou removida pelo administrador de sistema nessa interface Web, o código em PHP faz uma chamada REST ao núcleo do *framework*, que está sendo executado em Node.js, para informar que ele deve recarregar as configurações do banco de dados. Assim, ele reinicia suas operações liberando memória dos sensores e atuadores não mais necessários e carregando o código-fonte daqueles que devem entrar em atividade.

4.2.2 Criação de Atuadores e de Sensores

A primeira ideia que surge para facilitar a implementação dos sensores e atuadores é a de implementá-los como *plug-ins* usando a mesma linguagem de programação pela qual o *framework* foi desenvolvido, porém essa rigidez restringe a sua evolução e o seu uso por um administrador de sistema que possivelmente não a conhece, consequentemente lhe dificultando criá-los para atender às suas necessidades. Deve-se permitir, portanto, que exista a flexibilidade de escolha da linguagem de programação em que um componente será escrito. Para cumprir este objetivo, é necessário entender claramente como as mensagens são passadas entre cada componente, ou seja, do sensor para o *framework*, e dele para os atuadores.

Para transferir mensagens sem impor o uso de uma linguagem de programação específica, o *framework* Adapta usufrui basicamente do conceito de processos e dos fluxos padrões fornecidos pelo sistema operacional. Com eles, essa flexibilização torna-se possível, porque o *framework* monta o código-fonte que deve ser executado, inicia um processo do interpretador ou do compilador compatível passando o código-fonte para ele pela entrada padrão e, após o processo encerrar, ele analisa o estado final da execução pelo código de retorno[2] e coleta as informações expostas na saída padrão e na saída padrão de erro. Adotando-se esta estratégia, surge a flexibilidade quanto ao uso de diferentes linguagens de programação para a implementação dos componentes.

[2] Após cada processo ser executado, ele retorna um número inteiro capaz de resumir o estado resultante da execução. Por exemplo, em sistemas operacionais que adotam o padrão POSIX, qualquer número diferente de zero é interpretado como um encerramento com estado falho ou errôneo (IEEE. . . , 2018), enquanto zero representa êxito na execução.

Figura 5 – Exemplo de código-fonte de sensor escrito para Node.js

```
async function getrequesttime(url) {
    // carrega o módulo correto para tratar a requisição (HTTP ou HTTPS)
    var http = require(/https:\/\//i.test(url) ? 'https' : 'http'),
        result;

    await new Promise((proceed) => {
        // antes de iniciar a requisição, armazena o tempo
        var start = new Date();
        http.get(url, {
            agent: false,
            rejectUnauthorized: false
        }, () => {
            // após concluí-la, calcula a diferença de tempo
            result = new Date() - start;
            proceed()
        }).on('error', (error) => {
            // em caso de erro, imprima na saída padrão de erro
            // e encerre o processo com o código de retorno 1
            console.error(error);
            process.exit(1)
        })
    });

    // retorna em formato JSON
    return {duration: result}
}
```

A Figura 5 ilustra o código-fonte de um sensor escrito para Node.js que permite obter o tempo de requisição a uma determinada URL usando o protocolo HTTP. O *framework* faz uma chamada à função `getrequesttime` passando a URL desejada como parâmetro. Após realizar a chamada, o sensor retorna o objeto `{"duration": duracao_em_milisegundos}` para que a duração seja avaliada pelo manipulador das políticas de adaptação. Esse código-fonte seria inserido na tela da Figura 7. Nela, o administrador de sistemas insere um nome para identificá-lo, escolhe a linguagem de programação e entra com o seu código-fonte. Os campos para inserção do nome das funções ou dos métodos existentes no código e qual é o tipo de cada um de seus parâmetros são usados posteriormente para facilitar a manipulação das políticas de adaptação.

A Figura 6 mostra um fragmento que pode ser anexado automaticamente ao final do código-fonte base do sensor pelo *framework* durante a verificação da política de adaptação. Esse anexo será criado toda vez que uma chamada ao sensor precisar ser realizada e estará de acordo com os parâmetros informados na política de adaptação. Consequentemente, com o código-fonte completo pronto, o *framework* o passa para a entrada padrão do

Figura 6 – Exemplo de fragmento para ser anexado no final do código-fonte base do sensor, ilustrado na Figura 5, para chamar a função `getrequesttime`

```
/* O código do sensor na Figura 4 seria incluído aqui pelo framework. */

/* Abaixo, está o fragmento anexado ao código do sensor */
/* para realizar a chamada. */

(async () => {
    // armazena o resultado (no formato JSON)
    var response = await getrequesttime('http://www.url.com/servicotestado');

    // transforma em string o resultado que está no formato JSON
    var jsonstring = JSON.stringify(response);

    // imprime JSON em formato de string na saída padrão
    // para que o framework consiga recebê-lo
    console.log(jsonstring)
})()
```

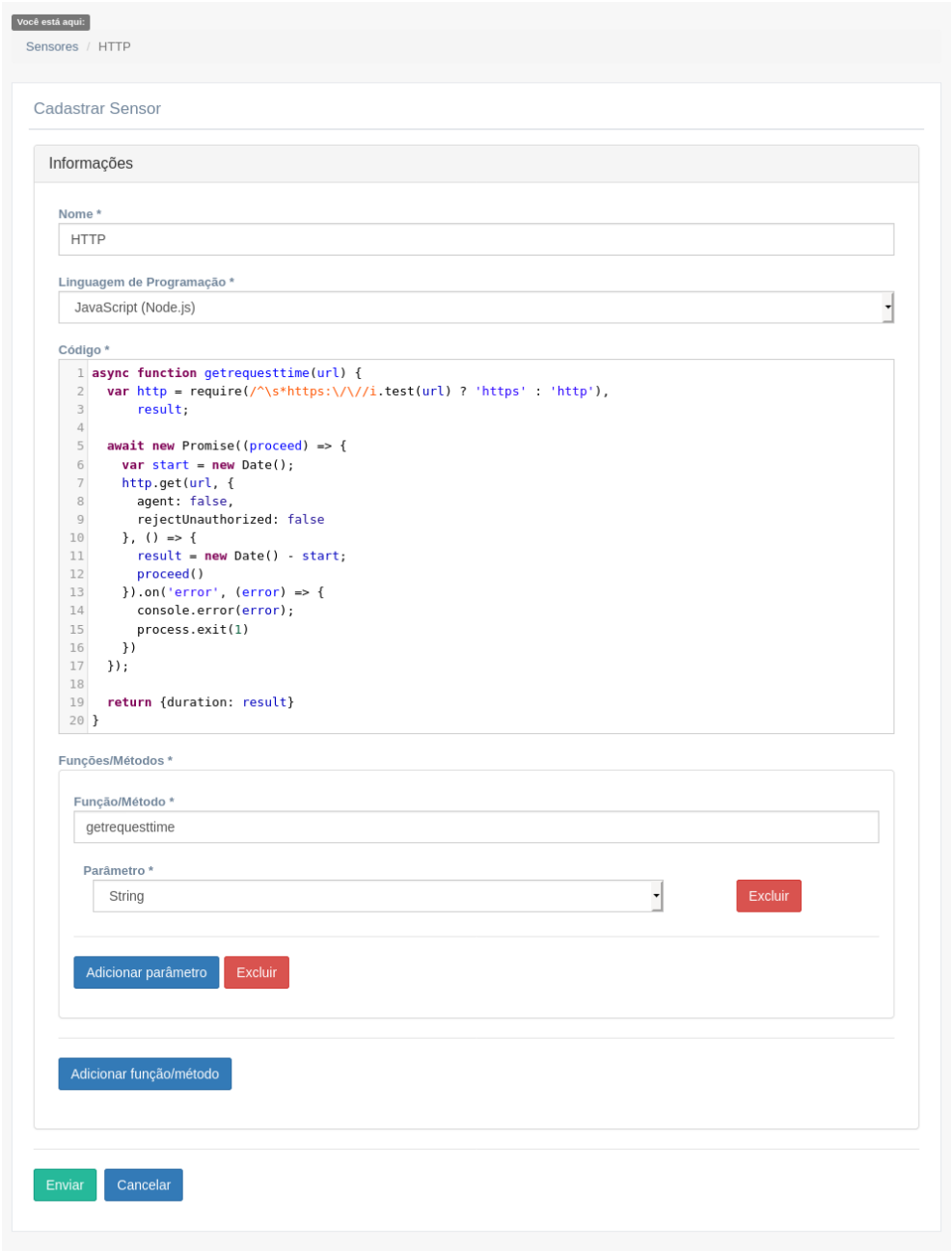
interpretador da linguagem de programação selecionada na tela de cadastro do sensor (no exemplo é Node.js). Ele é executado e obtém o tempo de requisição a uma determinada URL[3]. Após a execução, tanto os dados impressos na saída padrão quanto na saída padrão de erro serão capturados pelo *framework*. Então, os dados capturados em formato JSON serão novamente convertidos para objeto a fim de que sejam processados pelos manipuladores de políticas de adaptação.

A Figura 8 mostra a tela para cadastro do atuador. Assim como acontece na tela para cadastro do sensor na Figura 7, nela, o administrador de sistemas também insere uma identificação, escolhe a linguagem de programação, entra com o código-fonte, o nome das funções ou dos métodos existentes no código e seleciona qual é o tipo de cada um de seus parâmetros. No entanto, há uma particularidade nessa tela em relação àquela para cadastro do sensor: o primeiro parâmetro de cada função ou método sempre receberá dados em formato JSON. Uma ou mais decisões dos manipuladores das políticas de adaptação estão definidas nele e ele é sempre passado no primeiro parâmetro. Já os demais, a partir do segundo, sempre recebem os mesmos valores conforme definido pelo administrador de sistema na tela para definição da política.

Quanto à execução, os atuadores são executados de maneira similar aos sensores. Eles possuem um código-fonte base contendo a função ou o método responsável por aplicar uma política. Após ele ser cadastrado no *framework*, o administrador de sistema pode selecioná-lo para uso com base nas políticas de adaptação aceitas e informar a função ou

[3] No exemplo, a URL sendo testada na política de adaptação é "http://www.url.com/servicotestado".

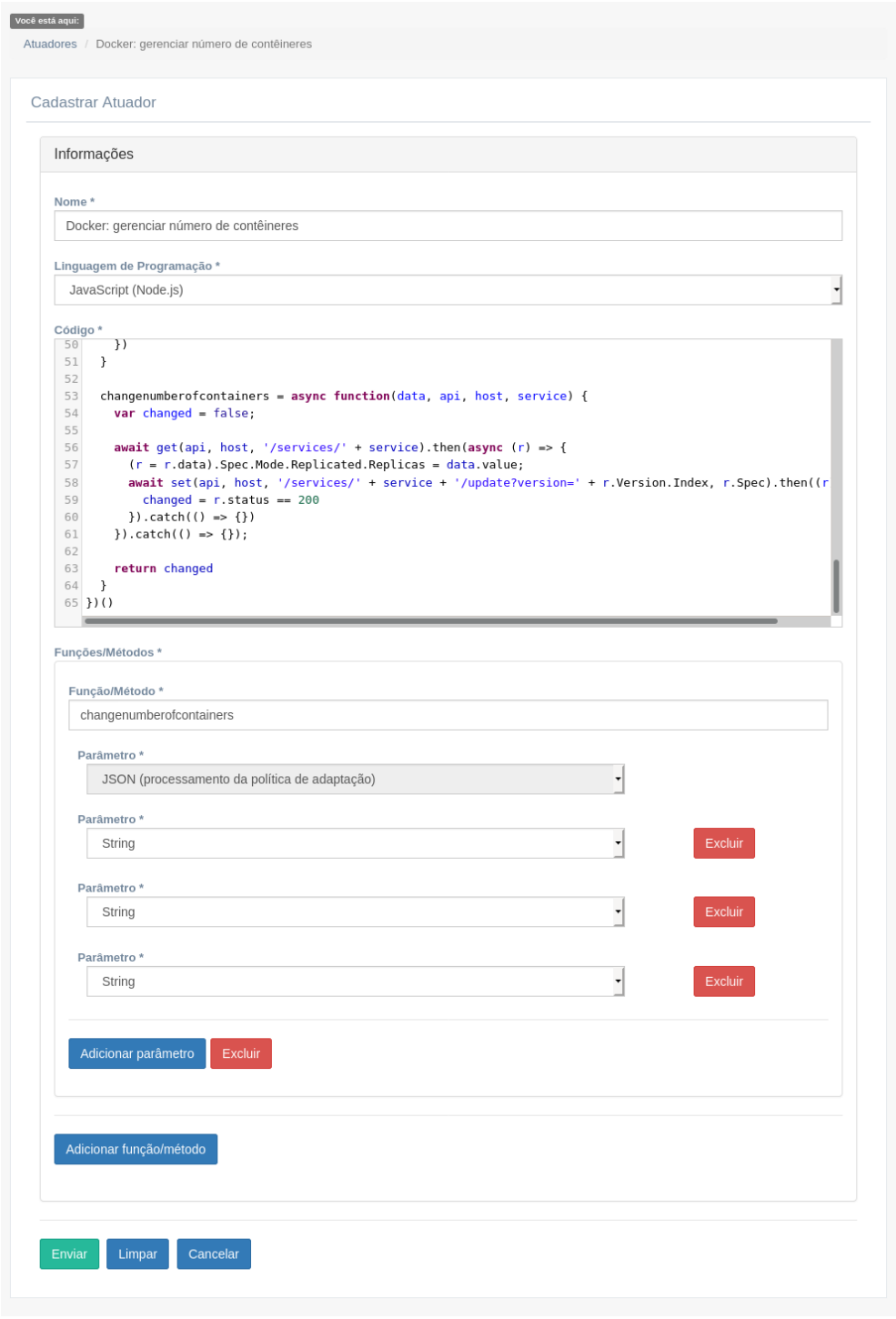
Figura 7 – Tela para cadastro de sensor



o método desejado. Por meio da entrada padrão, o *framework* passa o código-fonte que deve ser executado para o interpretador ou o compilador com os valores processados pela política de adaptação e os demais parâmetros necessários para o atuador operar. Então, ele inicia o processo de execução. Após concluir, o *framework* se informa sobre o estado final da execução do atuador pelo código de retorno do programa e obtém a descrição de falha pela saída padrão de erro.

Eis, portanto, o entendimento de que quanto mais abrangente o código do sensor e do atuador for, melhor será o seu aproveitamento. O comportamento será refinado com base

Figura 8 – Tela para cadastro de atuador



nos parâmetros informados na tela de configuração da política de adaptação. Por exemplo, o código-fonte do atuador pode ter compatibilidade com todas as versões de API do Docker e, com base na versão informada na política de adaptação, o atuador saberá como se conectar e se comunicar com o Docker na versão selecionada. Essa prática é importante, porque ela proporciona o reuso de qualquer componente do *framework* por qualquer

administrador de sistema em diferentes ambientes. À medida que mais componentes forem desenvolvidos e disponibilizados, os usuários poderão instalar um componente e apenas ajustar os parâmetros (tais como: endereço IP dos serviços usados, versão da API e os limites toleráveis) que atendem às suas necessidades de trabalho. Consequentemente, à medida que o tempo passar e mais componentes forem desenvolvidos, menos usuários precisarão implementar um novo componente para usá-lo de acordo com as suas demandas.

4.2.3 O Intercâmbio de Mensagens Padronizadas entre os Componentes

Quando um processo escreve dados na sua saída padrão, não há limitação de quantidade de informação que pode ser transferida para o outro que os receberá. Além disso, as informações são passadas sem sofrer qualquer modificação.

Como é prático escrever e ler dados em formato JSON (JavaScript Object Notation) graças à sua sintaxe simples (PEZOA et al., 2016), ela se torna atrativa para o intercâmbio de mensagens entre os componentes do *framework*. Outro detalhe importante está relacionado à quantidade de informações que pode ser passada por ela. Os componentes devem ter a liberdade de informar quantos parâmetros eles acharem necessários para que haja a tomada correta de decisão ou para que um atuador seja capaz de agir adequadamente quando apenas um único valor não for suficiente para descrever o que deve ser realmente feito.

Apesar das informações serem passadas de um processo para outro sem sofrer qualquer modificação, é necessário garantir que haja praticidade no formato transferido e que seja consumido a menor quantidade de bytes possível e tempo para processamento. Portanto, recomenda-se o uso do formato JSON (NURSEITOV et al., 2009). Assim, cada componente será capaz de emitir ou extrair, com facilidade, informações advindas de outro componente.

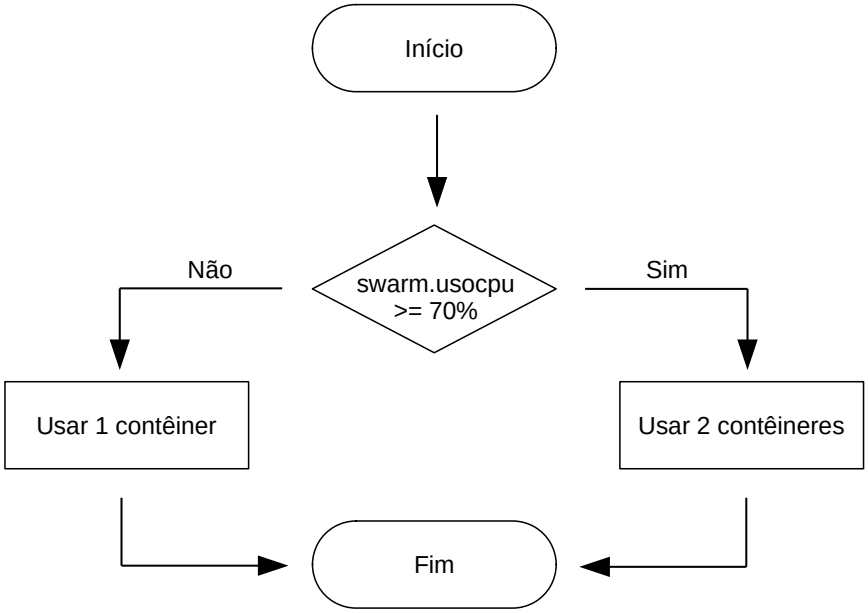
4.2.4 Manipuladores das Políticas de Adaptação

O *framework* passa os dados obtidos pelo sensor para o componente responsável pela tomada de decisão. Eles, então, devem ser confrontados com as políticas aceitáveis de adaptação em sistemas de *software* com capacidade autoadaptativa (FILIERI et al., 2017). As informações coletadas devem ser suficientes para que as melhores decisões sejam tomadas. Não pode haver escassez de dados, mas também não pode haver excessos. Por exemplo, os sensores não podem ocupar a memória e o processamento com informações irrelevantes para a tomada de decisões.

São os administradores de sistema que ajustam as políticas de adaptação. Para isso, além de fornecer os parâmetros aceitáveis de acordo com as necessidades do ambiente para o qual eles trabalham, eles devem selecionar o sensor e o atuador adequados para

cumprir uma determinada política.

Figura 9 – Política de adaptação em fluxograma ilustrando o ajuste do número de contêineres com base na carga de processamento



Assim como os sensores e os atuadores podem ser escritos em qualquer linguagem de programação, a mesma flexibilidade deve ser garantida para as políticas de adaptação. Enquanto há situações complexas que exigem uma análise minuciosa para que a melhor solução seja obtida, há outras mais simples que podem ser decididas com maior facilidade. Por exemplo, o *framework* pode ser configurado para ler o valor de uma propriedade coletada pelo sensor e, com base nele, orientar o comportamento dos atuadores. Neste caso, se o valor obtido for menor, igual ou maior em comparação a um determinado limite tolerável, então um atuador entrará em ação, senão ele não agirá ou poderá desempenhar outra atividade.

Por exemplo, a Figura 9 ilustra o *framework* controlando o número de contêineres ativos de um determinado serviço baseado nos dados coletados pelo sensor e passados no formato JSON para a política de adaptação. Entre esses dados, poderia haver a propriedade hipotética denominada "swarm.usocpu", que informa a porcentagem total de processamento consumida em todo o Docker Swarm por esse serviço. Se fosse definida uma política com limite tolerável de 70% para ocupação do processador, o atuador deveria interagir com o Docker Swarm por meio de sua API com o intuito de instanciar mais contêineres em outros dispositivos visando aliviar a carga entre as instâncias atuais.

5 Aplicação Prática

Para validar nossa abordagem para a introdução de capacidades autonômicas em sistemas legados, desenvolvemos cenários e experimentos em um sistema de *software* distribuído real. Apresentamos três cenários nesta seção, os quais lidam com a qualidade de serviço no sistema legado Moodle. Esse estudo demonstra que é possível e vantajoso aplicar nossa abordagem aos sistemas legados do mundo real.

5.1 Metodologia

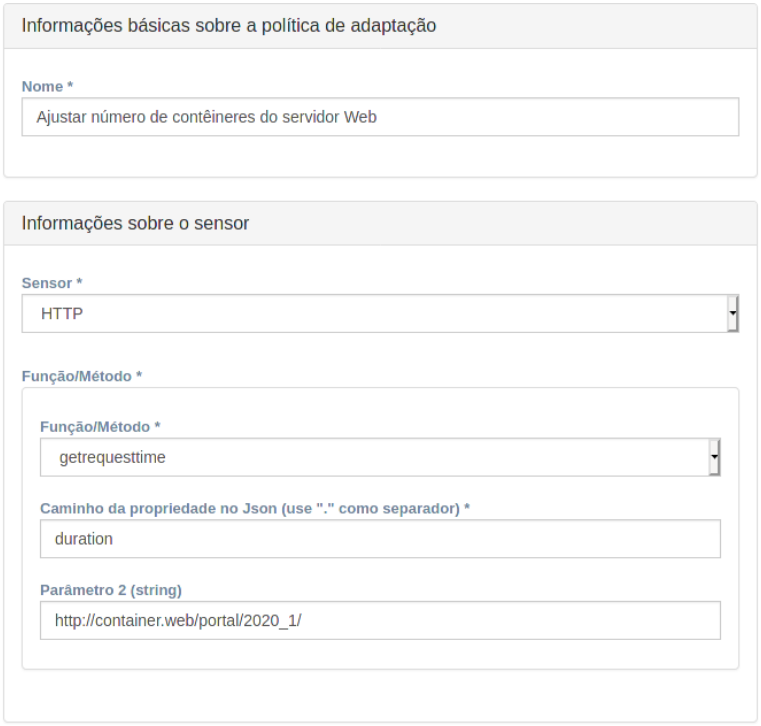
Avaliar a incorporação de recursos de computação autonômica é um processo complicado de engenharia de software. É preciso avaliar não apenas os efeitos causados pelos novos recursos de computação autonômica, mas também avaliar o comportamento do sistema original. Além do teste da eficácia e robustez em relação à incorporação de recursos de computação autônoma, é preciso adicionalmente avaliar em quais cenários os recursos de computação autonômica são realmente necessários e se é possível estabelecer uma configuração universalmente ideal e aplicá-la a todo o sistema de software. Com isso, projetamos uma avaliação de acordo com a seguinte metodologia:

Configuração do Ambiente de Execução. Aplicamos o *framework* a um sistema corporativo de larga escala, Moodle. Moodle é um sistema de código aberto, com milhões de linhas de código, em desenvolvimento ativo e mantido por um grande número de desenvolvedores de *software* espelhados no mundo. Milhares de organizações mundialmente dependem do Moodle para realizar suas atividades de educação a distância. Foi criada uma imagem da distribuição do GNU/Linux Ubuntu 18.04.2 para o Docker 18.09.4 contendo apenas o Servidor Web Apache 2.4.29 e a linguagem de programação PHP 7.2.15. O Apache foi executado no modo *prefork* para realizar o multi-processamento de requisições por meio de processos ao invés de usar o conceito de *threads* (como acontece nos módulos *event* e *worker*). Esse é o módulo tradicional de operação desse servidor Web e instalado como padrão nas distribuições GNU/Linux. O ambiente foi configurado em dois servidores com processador Intel Xeon E5620 com 2.40 GHz e 16 GB de memória RAM em rede local. O NFS foi usado para garantir o acesso a arquivos que devem ser compartilhados entre os contêineres (como aqueles arquivos contendo informações sobre a sessão de usuário) e a consistência deles (por exemplo, por meio de eventos mutuamente exclusivos – *locks* – diante de tentativas de acessos concorrentes quando, por exemplo, um arquivo estiver aberto e sendo lido por um processo em um contêiner e outro tentar abri-lo para modificá-lo, o que pode ocasionar problemas de leitura e inconsistências em sistemas distribuídos).

Configuração do Framework. Um sensor foi escrito para ser capaz de realizar

requisições a determinadas URLs definidas na política de adaptação e retornar o tempo de resposta (tela com o código completo para Node.js na Figura 7). Com base nele, as políticas de adaptação foram definidas visando reduzi-lo. Para modificar o número de contêineres, um atuador foi escrito visando interagir com o Docker Swarm por meio da sua API (tela com o código parcial para Node.js na Figura 8).

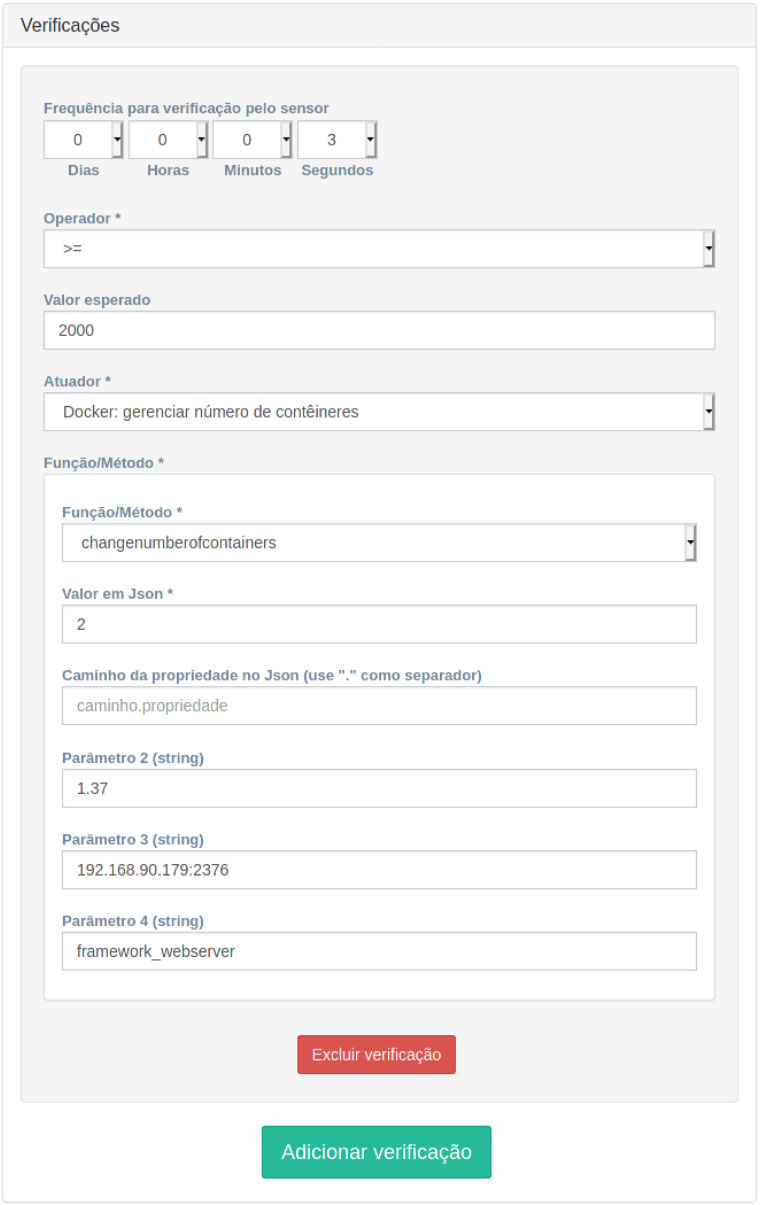
Figura 10 – Definição de qual sensor será usado, qual função será chamada e quais parâmetros serão passados.



O exemplo da Figura 10 ilustra a definição de uma política de adaptação. Nela, o sensor "HTTP" foi escolhido (definido na Figura 5). Se olharmos atentamente no código desse sensor, verificaremos que há uma função denominada `getrequesttime` implementada e que ela é responsável por retornar o tempo de resposta de requisição a uma URL definida na política de adaptação. Visando padronizar a comunicação entre os componentes do *framework*, os dados são retornados no formato JSON. Neste exemplo, uma das propriedades retornadas nele é denominada de `duration` e ela contém a informação que precisaremos para a situação observada.

Por outro lado, a Figura 11 mostra a configuração de uma verificação a ser feita pelo *framework*, qual é o limite tolerável e como ele deve responder. Na frequência de 3 segundos, ele requisita o recurso na URL informada por meio do sensor (Figura 10). Se o tempo de resposta for superior ou igual a 2000 milésimos (2 segundos), ele deverá reagir aumentando o número de contêineres que contém a imagem do software. Há diferentes tipos de operadores de comparação tais como `<` (quando for menor que), `>` (quando for maior que), `==` (quando for igual) etc. Para o cenário descrito, foi interessante definir que

Figura 11 – Definição de qual atuador será usado, qual função será chamada e quais parâmetros serão passados.



uma modificação ocorrerá quando o valor for maior ou igual ao limite definido. Portanto, o operador `>=` foi escolhido.

No exemplo, selecionamos o atuador "Docker: gerenciar número de contêineres" e a função denominada `changenumberofcontainers`, ambos definidos na Figura 8, e informamos o valor `2` no campo "Valor em Json" com o intuito de que ele responda aumentando para dois contêineres quando aquele limite de tempo de resposta exceder. O segundo parâmetro indica a versão da API usada pelo Docker, o terceiro indica o endereço IP e qual é a porta que o Docker espera receber conexões, e o quarto demonstra o nome do serviço que deverá ter o seu número de contêineres ajustado. É importante relembrar que a ordem, o tipo e a necessidade de preenchimento desses e de outros campos podem

variar dependendo da implementação do atuador. O campo "Caminho da propriedade no Json" foi deixado em branco, porque, quando preenchido, o *framework* coloca o valor inserido no formato Json obedecendo o caminho informado (por exemplo, se ele fosse preenchido com "duration", então o *framework* passaria, para a função, o parâmetro Json `{"duration": 2}` com outras propriedades geradas pela política de adaptação ao invés de apenas o valor `2`) já que esse formato pode ser necessário para algumas implementações de atuadores e outras informações podem ser importantes para a atividade do atuador.

Métricas. Para avaliar este trabalho, usamos o tempo de resposta como métrica. O tempo de resposta é a quantidade de tempo entre uma requisição ser recebida pelo servidor, ser processada e a resposta ser emitida de volta para a parte requisitante. É o registro completo de quanto tempo leva para que o sistema processe a transação solicitada. Tempo muito elevado pode indicar: incapacidade para tratar uma quantidade excessiva de requisições recebidas durante um intervalo de tempo; ou incidentes na rede (por exemplo, perda de pacotes com tráfego excessivo). Sem dúvidas, foi esperado que, diante do aumento gradual do número de requisições, o sistema deveria manter o tempo de resposta menor possível e não apresentar falhas, ou seja, ele não poderia deixar de responder a uma requisição por ficar temporariamente inoperante devido ao atingimento da capacidade máxima de memória RAM ou processamento.

Método de Condução dos Experimentos. Os testes foram conduzidos usando Locust, que é uma ferramenta completamente baseada em Python para a realização de testes de carga. Este tipo de teste é muito importante para a engenharia, porque ele testa os limites do *software* e avalia seu comportamento, ou seja, visa identificar quanta carga ele suporta. Então, Locust simula usuários *on-line* e identifica quantos acessos um sistema consegue atender, qual é o tempo médio de resposta e qual é o número de requisições não atendidas devido a falhas. *Scripts* são escritos para a realização dos testes em uma determinada URL e as atividades são monitoradas em tempo real, o que facilita a identificação de gargalos no sistema. Além desse acompanhamento, Locust permite obter arquivos no formato CSV, que contêm informações sobre a condução dos testes. Esse método foi usado para realizarmos a análise comportamental dos Servidores Web em funcionamento nos contêineres e do número de instâncias controladas por meio de reações do *framework*.

Para cada um dos três cenários, executamos os testes de carga via Locust para uma carga máxima de 50, 100, 300, 500, 700 e 1000 usuários. Cada experimento foi configurado para executar por 10 minutos. A maneira como Locust realiza as requisições ao Servidor Web fez os gráficos apresentarem uma súbita elevação do tempo de resposta nas requisições iniciais. À medida que os testes ocorrem, o tempo de resposta tende a cair.

5.2 Cenários

5.2.1 Cenário I - Página Principal

Descrição: A página principal produzida pelo Moodle é relativamente simples: ela basicamente lista as disciplinas associadas ao usuário autenticado no sistema. Do ponto de vista de acesso a recursos do sistema, consideramos que este cenário representa um baixo índice de acesso ao SGBD e ao sistema de arquivos. Com isso, consideramos este um cenário básico. Esperamos que a implantação tradicional não apresente uma sobrecarga considerável no tempo de resposta, mesmo em casos com um grande volume de acessos simultâneos (iguais a 1000 usuários por segundo).

Resultados: Como podemos observar na Tabela 2, o tempo de resposta médio variou entre 161 e 18.766 ms para a implantação tradicional e entre 215 e 11.878 ms para a implantação com o *framework* Adapta. Para um volume médio de acesso simultâneo maior que 500, a implantação com o *framework* Adapta apresentou uma redução de 33% no tempo de resposta. Já para o número de total de falhas, a implantação com o *framework* Adapta apresentou reduções significativas como podemos observar na Tabela 3. No pior caso, a implantação tradicional apresentou uma taxa média de 80 falhas, enquanto a implantação com o *framework* Adapta apresentou uma taxa média máxima de 3 falhas.

Tabela 2 – Cenário I - Estatística Descritiva - Tempo de Resposta

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	1297	161	218.3
100	0	1336	180	251.2
300	0	4292	599	916
500	0	8675	5242	5364
700	0	13605	10497	10038
1000	0	19783	18766	16355
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	1776	215.0	288.9
100	0	1679	197.0	258.3
300	0	2532	425.0	624.3
500	0	4037	3938	3661
700	0	7825	7071	6085
1000	0	13586	11878	10103

Tabela 3 – Cenário I - Estatística Descritiva - Número de Falhas

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	0	0	0
300	0	1	0	0.3056
500	0	9	9.000	8.226
700	0	13	13.00	11.58
1000	0	80	58.0	47.5
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	0	0	0
300	0	3	2	1.664
500	0	1	1	0.9367
700	0	3	3	2.76
1000	0	0	0	0

Discussão: De fato, como podemos observar na Figura 12, a implantação tradicional apresentou um tempo médio satisfatório e uma taxa de falhas relativamente baixa, mesmo com um grande volume de acessos simultâneos (em média 20.000 ms com 1000 usuários por segundo). Do mesmo modo, podemos observar na Figura 12 que o

Figura 12 – Cenário I - Tempo Médio de Resposta por Volume de Acessos Simultâneos

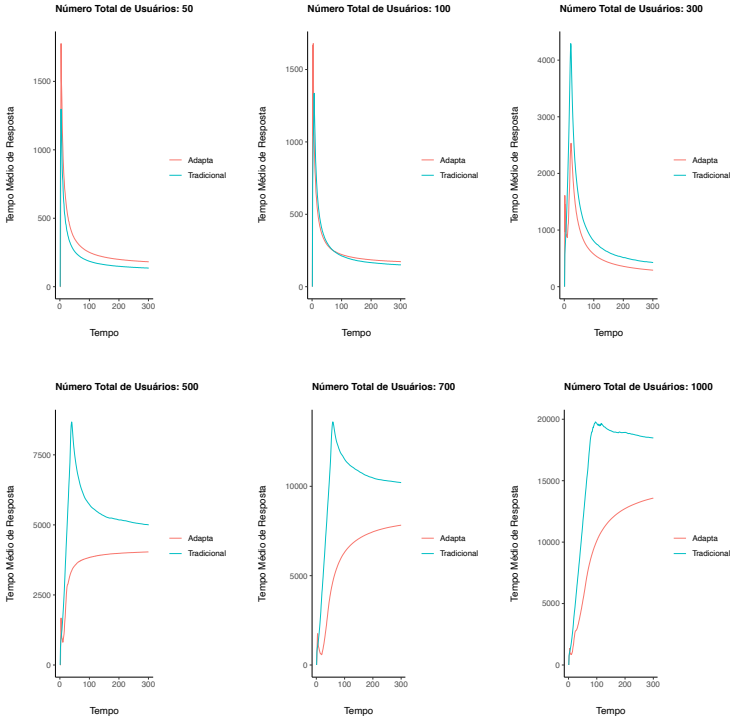
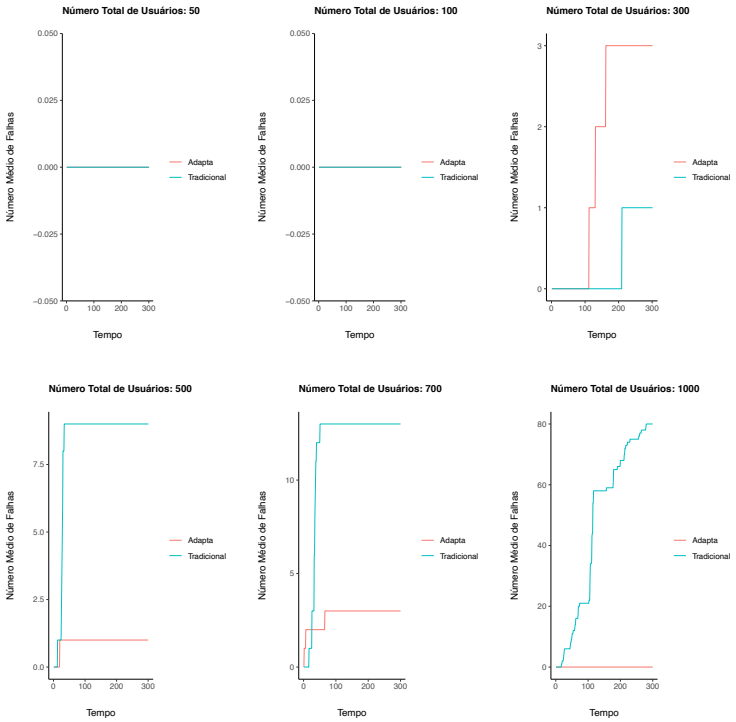


Figura 13 – Cenário I - Taxa Média de Falhas por Volume de Acessos Simultâneos



uso da computação autonômica por meio do *framework* Adapta só apresentou ganhos significativos com um volume médio de acessos simultâneos maior que 500 usuários por segundo. Com isso, podemos destacar que a implantação do *framework* Adapta é vantajosa mesmo quando consideramos um ambiente variando de dois a quatro contêineres em cenários básicos nos quais o índice de acesso ao SGBD e ao sistema de arquivos é baixo. Do ponto de vista da taxa média de falhas (veja Figura 13), podemos perceber que o *framework* Adapta auxiliou a manter o atendimento às requisições, isto é, uma taxa média de falhas bem próxima de zero.

5.2.2 Cenário II - Página de Administração

Descrição: Diferentemente da página principal, a página de administração requer que os principais arquivos de configuração dos módulos ativos sejam percorridos pelo Moodle. Consideramos este cenário como intermediário, pois apresenta um alto índice de acesso ao sistema de arquivos e um índice médio de acesso ao SGBD. É importante observar que o acesso aos arquivos pode demandar um tempo de resposta elevado quando a implantação do sistema estiver sobre um sistema remoto de arquivos (e.g., NFS). Além disso, o Moodle faz diversas consultas ao SGBD sobre permissões de acesso e configurações dos *plug-ins*, o que eleva a demanda por acesso ao SGBD.

Tabela 4 – Cenário II - Estatística Descritiva - Tempo de Resposta

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	1788	346.0	424.9
100	0	2742	470.0	614.7
300	0	10216	9624	9239
500	0	20533	20179	18164
700	0	31234	30004	25777
1000	0	43545	39830	33037
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	4462.0	600.0	818.5
100	0	2884.0	538.0	638.0
300	0	3114.0	929.5	1199.0
500	0	4818	4780	4517
700	0	9185	8738	7817
1000	0	15131	13470	11521

Tabela 5 – Cenário II - Estatística Descritiva - Número de Falhas

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	0	0	0
300	0	0	0	0
500	0	3	3	2.664
700	0	3	3	2.64
1000	0	416	179	174.7
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	0	0	0
300	0	1	1	0.49
500	0	6	5	5.19
700	0	2	2	1.893
1000	0	2	2	1.99

Resultados: Para um cenário mais complexo, como podemos observar na Tabela 4, o tempo de resposta médio apresentou uma variação considerável, variando entre 346 e 39.830 ms para a implantação tradicional e entre 600 e 13.470 ms para a implantação com o *framework* Adapta. Neste cenário, a implantação com ele apresentou uma redução real

Figura 14 – Cenário II - Tempo Médio de Resposta por Volume de Acessos Simultâneos

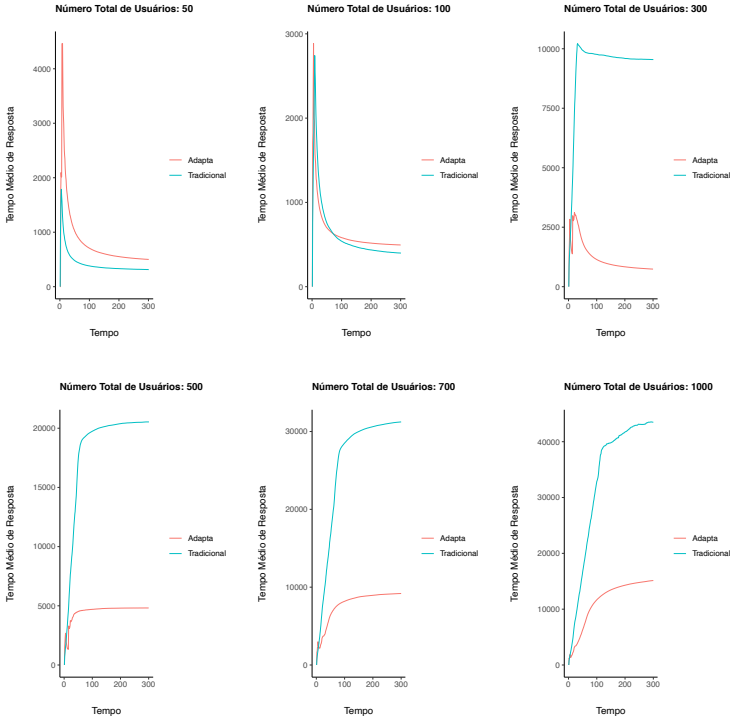
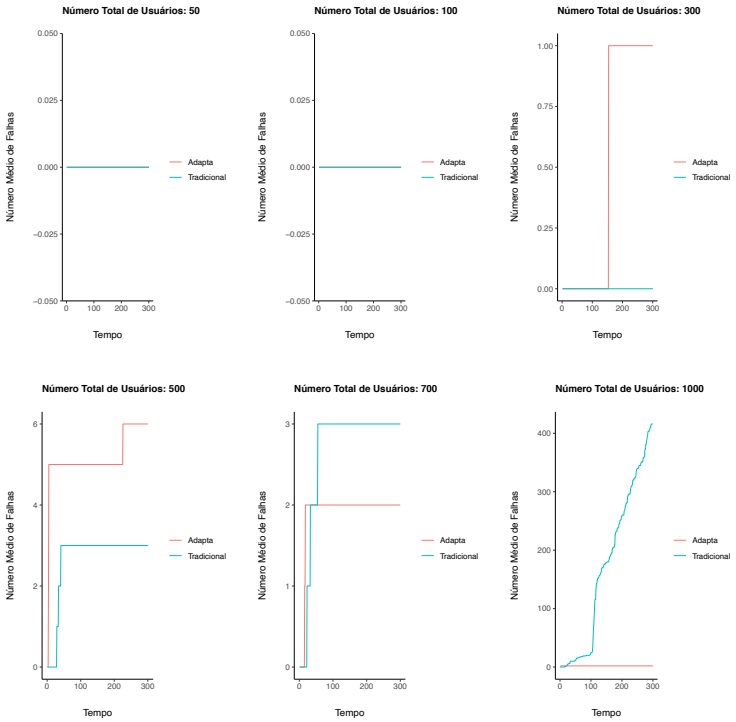


Figura 15 – Cenário II - Taxa Média de Falhas por Volume de Acessos Simultâneos



e considerável de aproximadamente 66% no tempo de resposta. Considerando o número de total de falhas, ele manteve o comportamento observado no Cenário I, apresentando reduções significativas como podemos observar na Tabela 5. Para o Cenário II, a implantação tradicional apresentou uma taxa média elevada de falhas, sendo 179 no pior caso, enquanto a implantação com o *framework* Adapta manteve uma excelente taxa média de 6 falhas no pior caso.

Discussão: Como podemos observar na Figura 14, para um cenário mais avançado, uma implantação do Moodle com base no *framework* Adapta apresentou ganhos significativos no tempo de resposta a partir da taxa média de acessos simultâneos de aproximadamente 300 usuários por segundo. O tempo médio de resposta permaneceu relativamente estável com uma sobrecarga no número de usuários, enquanto que com a implementação tradicional, o tempo de resposta cresceu abruptamente em relação à sobrecarga no número de usuário. Do ponto de vista do número de falhas (veja Figura 15), podemos observar o melhor e mais indicado comportamento que o *framework* Adapta apresenta. Mesmo com uma sobrecarga de 1.000 usuários por segundo, o *framework* conseguiu com sucesso manter a taxa de falhas bem próxima de zero.

5.2.3 Cenário III - Módulo Fórum

Tabela 6 – Cenário III - Estatística Descritiva - Tempo de Resposta

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	1968	499	577
100	0	3947	3030	3064
300	0	13231	13123	12213
500	0	22995	22052	19138
700	0	32382	29584	24539
1000	0	41443	35440	29379
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	1185	410.0	448.1
100	0	1629	542.0	605.9
300	0	4117	4052	3845
500	0	8025	7659	6927
700	0	11773	10859	9540
1000	0	17017	14899	12717

Tabela 7 – Cenário III - Estatística Descritiva - Número de Falhas

Implantação Tradicional				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	0	0	0
300	0	285	118.5	126.2
500	0	317	148.5	141.59
700	0	232.0	100	101.3
1000	0	987.0	312.5	385.0
Implantação *Framework* Adapta				
Num. Usuários	Min	Max	Median	Mean
50	0	0	0	0
100	0	1	1	1
300	0	1	1	0.9367
500	0	1	1	0.9799
700	0	1	1	0.9666
1000	0	1	1	0.9431

Descrição: O módulo Fórum é aplicado para organizar grandes discussões nas quais o professor pretende acompanhar a participação e a interação entre os alunos. Este é um cenário considerado complexo, pois envolve a execução de grande volume de consultas e persistências de dados no SGBD, apesar de não apresentar um volume considerável de acesso ao Sistema de Arquivos.

Figura 16 – Cenário III - Tempo Médio de Resposta por Volume de Acessos Simultâneos

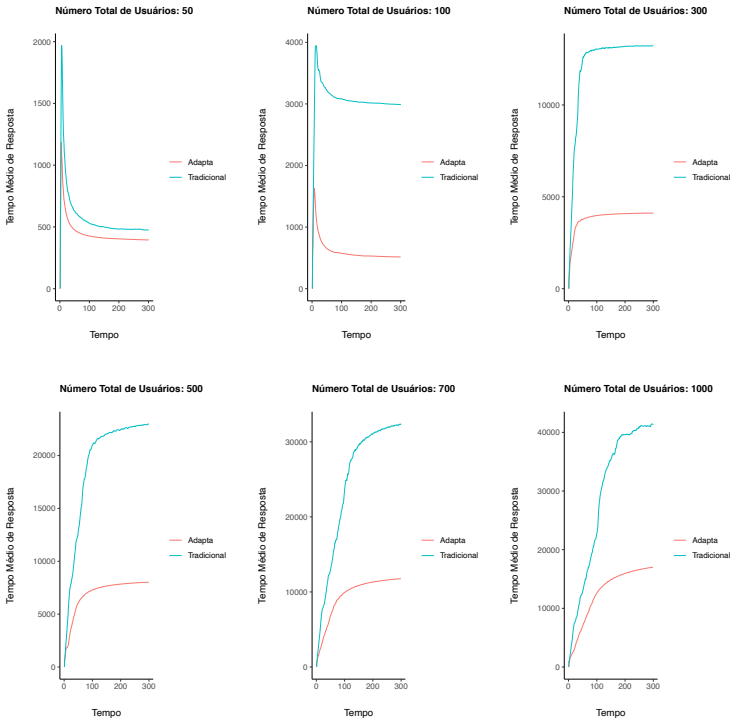
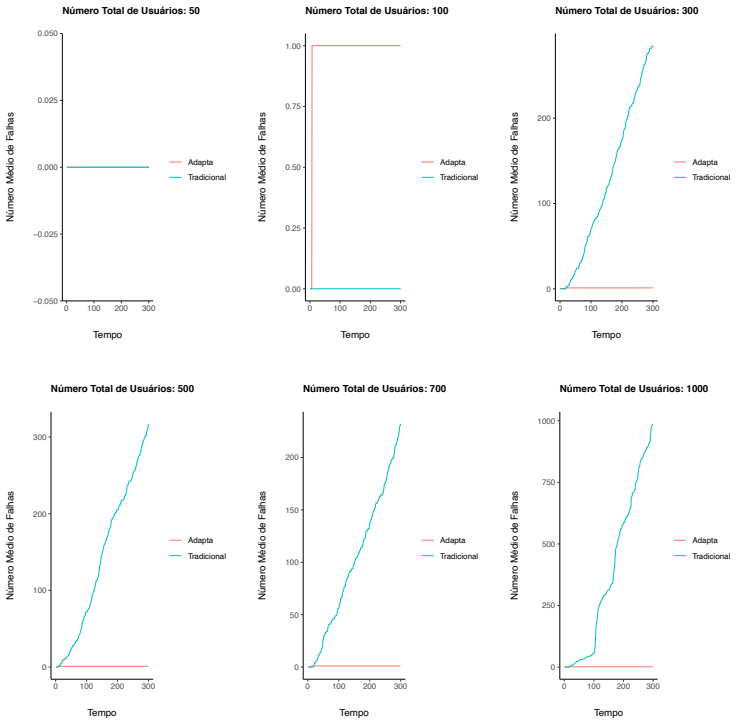


Figura 17 – Cenário III - Taxa Média de Falhas por Volume de Acessos Simultâneos



Resultados: Para um cenário onde o volume de acesso ao SGBD predomina, o tempo de resposta médio apresentou uma variação considerável e cresceu rapidamente no caso da implantação tradicional, variando entre 499 e 35.440 ms (veja Tabela 6). Para a implantação com o *framework* Adapta, o tempo médio de resposta variou entre 410 e 14.899 ms. Neste cenário, ele apresentou um ganho na redução do tempo de resposta similar ao ganho no Cenário II. Considerando o número de total de falhas, ele manteve o comportamento observado nos cenários anteriores, apresentando reduções significativas como podemos observar na Tabela 7. Para o Cenário III, a implantação tradicional apresentou uma taxa média ainda mais elevada de falhas, sendo uma taxa média de 312,5 falhas no pior caso, enquanto a implantação com o *framework* Adapta manteve uma excelente taxa média de falhas.

Discussão: Pela Figura 16, podemos observar que, mesmo para um número reduzido de acessos simultâneos, em torno de 100, a implantação com o *framework* Adapta apresentou melhoras significativas no tempo de resposta para o Cenário III. Com o gatilho automático de autoadaptação, o tempo de resposta despencou de uma média de aproximadamente 3.000 ms para aproximadamente 700 ms, uma taxa em torno de 4 vezes menor. Podemos observar que, do mesmo modo para os demais cenários, o *framework* Adapta foi capaz de manter a estabilidade do tempo de resposta, ao contrário da implantação tradicional, cujo tempo médio de resposta cresceu drasticamente ao longo do tempo à medida que o número de usuários simultâneos cresceu (veja Figura 16). Do mesmo modo, em relação aos demais cenários, o *framework* Adapta manteve uma taxa de falhas bem próxima de zero (veja Figura 17), o que reforça a nossa percepção positiva da capacidade do *framework* em introduzir capacidades autonômicas em sistemas legados.

6 Discussões e Lições Aprendidas

A modularidade é a principal característica da nossa abordagem. Ela possibilita que os sensores e os atuadores interajam com o recurso monitorado ou com o ambiente de suporte ao *software* por meio de diferentes APIs independentemente de uma linguagem de programação específica.

Abaixo, há três exemplos observados para os quais o *framework* também traz benefícios e com os quais ele poderia ser integrado futuramente: interação com dispositivos que usando o protocolo SNMP (seção 6.1); adaptação das políticas de uma rede SDN (seção 6.2); e o monitoramento direto do *software* por meio da Programação Orientada a Aspectos (seção 6.3).

6.1 Reparação de Serviços na Rede usando SNMP

Sem dúvidas, é necessário conhecer o estado de cada um dos componentes da rede para administrá-la bem e integralmente. Como ela é composta por dispositivos de diferentes fabricantes, havia, portanto, a necessidade de criar uma linguagem padrão para permitir o gerenciamento de recursos de rede. Então, surge o Protocolo Simples para Gerenciamento de Rede (SNMP) (CASE et al., 1990), que se tornou um padrão e é apoiado pela grande maioria de fabricantes de equipamentos. O resultado é um protocolo que possui simplicidade na sua implementação e que inclui um conjunto mínimo de comandos, mas que lhe permite gerenciar quase todas as tarefas dos dispositivos de rede.

Todas as informações necessárias para gerenciar os equipamentos são armazenadas na sua própria Base de Informações Administrativas (MIB). Essa base é formada por um conjunto de variáveis que caracterizam o estado do objeto monitorado. Elas refletem o espaço disponível por partição, percentual de uso de CPU, número de processos ativos, quantidade de memória RAM usada e disponível, número de pacotes processados pelo dispositivo, o estado de suas interfaces de rede etc. Além dessas variáveis padrões, cada fabricante de equipamentos inclui, na MIB, alguns parâmetros específicos para os seus dispositivos. Cada dispositivo gerenciado é um nó que implementa uma interface SNMP visando permitir acesso unidirecional (somente leitura) ou bidirecional (tanto leitura quanto modificação) de suas informações específicas. Eles trocam dados com uma Estação de Gerenciamento de Rede (NMS), que executa aplicações capazes de controlar e gerenciar os dispositivos gerenciados.

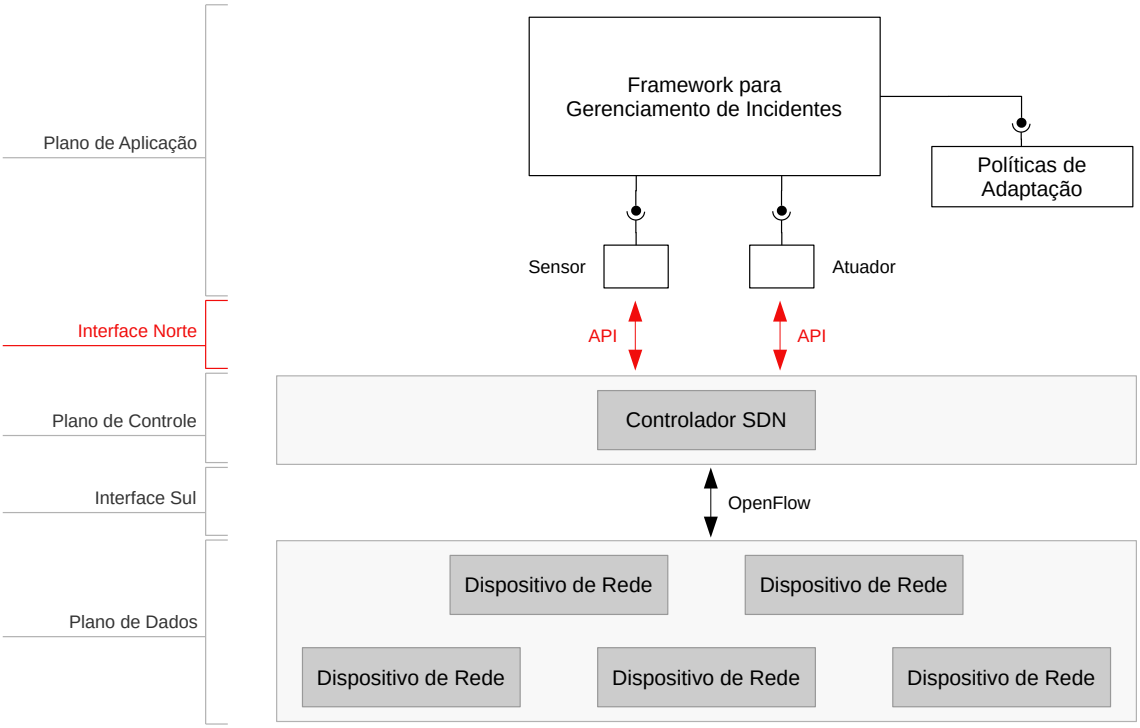
Como muitos dispositivos já o implementam, ele pode ser usado pelo *framework* como um método para supervisionar e administrar a rede assim como em (ROOHI;

RAEISIFARD; IBRAHIM, 2014) e em (AFFANDI et al., 2015), e, portanto, acelerar a adoção do *framework*. A observação da operação do SNMP também nos deixa um conceito interessante: agente, que é um programa executado no dispositivo gerenciado para ler as suas informações relevantes e, consequentemente, enviá-las para a NMS. Além do mecanismo de busca de dados, ele também fornece um mecanismo proativo, chamado `Trap`, que notifica a NMS quando determinadas condições são atendidas (por exemplo: erros no sistema que exigem ações urgentes).

6.2 Adaptação da Política de um Controlador SDN

As redes de computadores tradicionais têm deficiências que precisam ser superadas, porque elas nem sempre são capazes de responder adequadamente às necessidades atuais de Tecnologia da Informação (KARAKUS; DURRESI, 2017). Elas são hierárquicas e a sua arquitetura estática atende inadequadamente às necessidades dinâmicas de computação e armazenamento atuais. Algumas das principais tendências que definem a necessidade de um novo paradigma incluem: o aumento do tráfego de dados; o crescente número de serviços em nuvem; o desenvolvimento da Internet das Coisas; construção e teste de novos serviços para as redes de computação; etc.

Figura 18 – Atuador e sensor do *framework* interagindo com a API do controlador SDN



SDN (Software Defined Network) é um novo paradigma de rede no qual a arquitetura deixa um modelo totalmente distribuído (cada dispositivo possui regras de operação independentes dos demais) para adotar uma abordagem mais centralizada (o tráfego de

dados entre os dispositivos passa a ser autorizado ou bloqueado por regras determinadas por um dispositivo central). Esta abordagem é caracterizada pela separação entre o plano de dados e o plano de controle (KARAKUS; DURRESI, 2017). O plano de dados inclui os elementos que realizam o encaminhamento de dados (*switches* e roteadores), e o plano de controle inclui o controlador, que proporciona um alto nível de abstração para a realização do controle dos elementos de encaminhamento no plano de dados. Então, o controlador passa a ser responsável por aplicar as regras estabelecidas para gerenciar a rede. O seu papel é crucial e, dependendo da sua maneira de operação, ele pode ocasionar o sucesso ou o fracasso da arquitetura de SDNs.

A rede é gerenciada por meio das aplicações, no plano de aplicação, que interagem com os controladores. As mensagens trocadas entre o plano de controle (do controlador) e esse plano de aplicação é realizada usando a denominada "interface norte". Como ainda não há um padrão desenvolvido para ela, cada controlador usa a sua própria. Já a permuta de mensagens entre o plano de controle e o plano de dados ocorre por meio da denominada "interface sul" que usa, por exemplo, o OpenFlow (protocolo de comunicação mais popular).

Portanto, há relevante interesse na capacidade autoadaptativa em redes SDN também (DARABSEH et al., 2015) (AL-AYYOUB et al., 2015) (TOMOVIC; PRASAD; RADUSINOVIC, 2014). Então, conforme a Figura 18 ilustra, é exatamente no plano de aplicação que entra o papel do *framework* com seus atuadores visando interagir com o controlador SDN por meio da sua API na interface norte e realizar a adaptação necessária na rede. Não apenas atuadores, mas sensores também podem ser usados para buscar informações sobre a situação atual da arquitetura. Após confrontar as informações coletadas sobre a atual situação da rede com as políticas de adaptação, o *framework* instrui os controladores por meio da sua API sobre como a rede deve adaptar-se para atender as novas demandas. Por exemplo, sempre que houver uma sobrecarga nela, o gargalo poderá ser resolvido causando uma mudança na infraestrutura em prol de um *software* operando sobre ela sem afetar a transparência para os usuários.

Para evitar a reescrita de atuadores para lidar com cada tipo de controlador disponível (ONOS, por exemplo), o *plug-in* poderá ser escrito uma vez, tornar-se público e ser importado no *framework* sempre que um administrador precisar usá-lo para interagir com os controladores. A sua reusabilidade é atrativa por propiciar que diversos atuadores e sensores sejam disponibilizados para diferentes tipos de controladores tais como ONOS, Onix, Trema, Floodlight e OpenDaylight. Devido à reusabilidade e modularidade, ter *plug-ings* já disponíveis para tratar uma tecnologia particular torna mais fácil e ágil para um administrador configurá-lo para o seu ambiente em prol de um *software* crítico operando nele. Neste caso, se uma infraestrutura de rede que usa SDNs necessitar ser readaptada para atender uma variação na demanda operacional, um *plug-in* já implementado e capaz de realizar essa interação entre o *framework* e os controladores será importado e os parâmetros

serão apenas ajustados para que ele colabore com a adequada acomodação do sistema nesse ambiente.

6.3 Programação Orientada a Aspectos

O objetivo do desenvolvimento de software é modelar parte do fluxo de informação que ocorre no mundo real, e as técnicas de sua engenharia propõem a implementação de sistemas de *software* decompostos em módulos. A Programação Orientada a Objetos (POO) modular melhora a reusabilidade, a extensibilidade e a legibilidade dos sistemas de *software*. O ponto central dessa abordagem está na definição de quais objetos devem servir como unidades primordiais de um módulo e a consequente associação deles para trazer o íntegro comportamento do sistema.

Porém, quando se cria sistemas de *software*, surgem muitas outras necessidades para abordar monitoramento de desempenho, registros de eventos, verificações de segurança, tratamento de exceções etc. Tradicionalmente, o desenvolvedor mistura essas necessidades com a lógica de negócio de cada módulo em diferentes partes do *software*, o que enfraquece o propósito da POO, porque a reusabilidade dos módulos pode ser grandemente impactada, e a manutenção e a reusabilidade do sistema serem limitadas.

O propósito da Programação Orientada a Aspectos (POA) é separar essas atividades (registro de eventos, o controle de segurança, a coleta sobre estatísticas de desempenho etc) do código da lógica de negócio. Ao separá-los, ela as torna independentes a fim de que a mudança de seus comportamentos não afete a lógica de operação e, posteriormente, para realizar a compilação, elas são recombinadas usando uma técnica chamada de *Weaving* para gerar um executável final (IRWIN et al., 1997) (ALEXANDER; BIEMAN; ANDREWS, 2004).

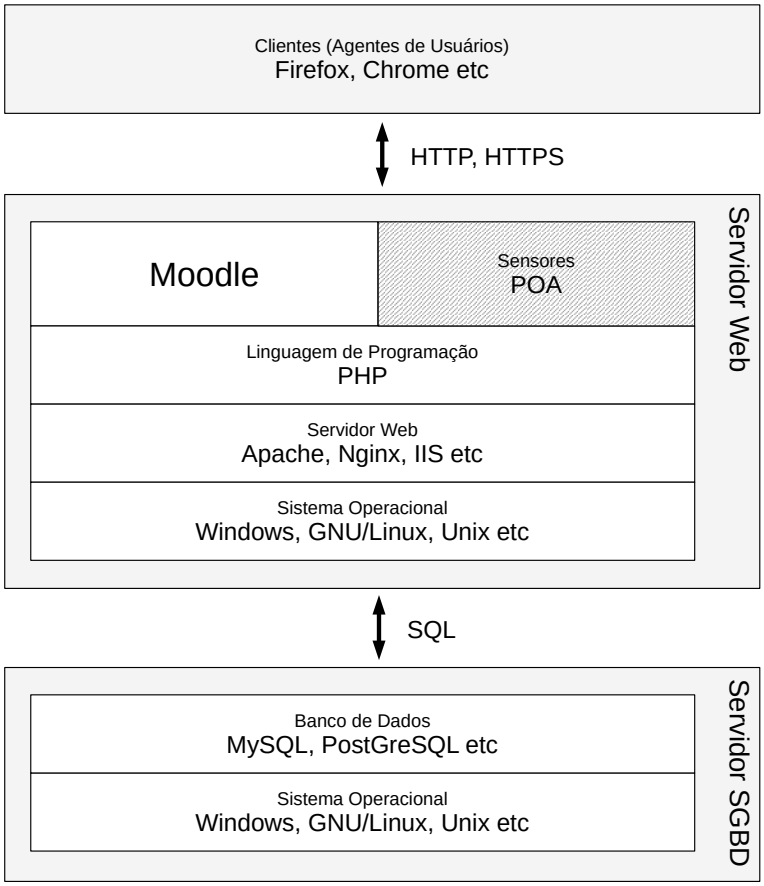
Alguns conceitos básicos da POA são:

- *Target:* um objeto é chamado de *target* se a sua execução for modificada pela POA;
- *Joinpoint:* define um ponto específico no programa onde a nova lógica será adicionada. Pontos típicos são: inicialização do objeto, chamada a um método e assim por diante;
- *Advice:* código a ser acrescentado;
- *Pointcut:* conjunto de regras que definem quando um *advice* será executado. Por elas, podemos controlar exatamente qual componente do programa recebe qual *advice*;
- *Aspect:* combinação de um *advice* com um *pointcut*. O aspecto define a lógica que deverá ser incluída em uma parte do programa e quando ela será executada; e

- *Weaving:* processo de adicionar os aspectos ao código seja durante a compilação (estática) ou a execução (dinâmica).

É por meio de um aspecto (combinação de *advice* com *pointcut*) que o *framework* pode obter informações sobre o recurso gerenciável. Ao longo do código fonte do recurso gerenciável, alguns aspectos ficam ligados a funções e métodos cruciais. Quando invocações acontecem, a POA as intercepta para gerar estatísticas e outros dados relevantes para que o *framework* as compare com as políticas de adaptação aceitáveis.

Figura 19 – POA permite obter informações diretamente do *software*



A Figura 19 ilustra informações sobre o sistema sendo obtidas por meio da utilização de técnicas de POA escritas para uma linguagem de programação e executando paralelamente com o sistema a fim de agir gerando informações para que o *framework* reaja. Por exemplo, ela pode identificar um aumento na frequência de chamada a métodos críticos do *software*. Se uma condição for satisfeita pelas políticas de adaptação com base nas informações coletadas pelos sensores, uma reação será tomada no sentido de que ocorra uma adaptação adequada para o sistema. Ele pode reagir automaticamente a determinados eventos emergenciais seja replicando as instâncias da aplicação para assegurar seu desempenho tolerável diante do aumento do número de requisições ou readaptando a

infraestrutura SDN para cumprir as políticas de adaptação. Para estes casos, um aspecto é criado para interceptar as invocações a uma função específica do sistema a fim de observar uma iminente sobrecarga e informar os dados relevantes para o *framework* com o intuito de que ele aja com base nessa política pré-definida.

7 Considerações Finais

O principal propósito da computação autonômica é permitir que um sistema de informação se gerencie e mantenha sua disponibilidade por meio de tecnologias adaptativas que aumentam significativamente o seu poder de computação e reduzem o tempo necessário que profissionais de tecnologia da informação levam para solucionar incidentes e realizar outras manutenções.

Esta dissertação descreve a experiência de escrever e experimentar um sistema que se encaixa entre essas tecnologias adaptativas. Ele foi desenvolvido de forma modular com: sensores, capazes de obter informações sobre o ambiente de implantação de um sistema; atuadores, capazes de realizar mudanças no ambiente segundo as políticas de adaptação definidas; e flexibilidade para permitir a escrita de novos componentes usando diversas linguagens de programação.

O experimento usou o Ambiente Virtual de Aprendizagem Moodle em uma rede com dois servidores integrados por meio de Docker Swarm. Embora esse recurso de conteinerização fosse usado ao invés de testá-lo diretamente sobre uma plataforma de serviços de computação em nuvem tais como Amazon Web Services (AWS), Microsoft Azure, Google Cloud Plataform etc, precisávamos avaliar previamente o seu comportamento e conseguimos observar reações positivas do *framework* Adapta diante da necessidade de mudanças no ambiente de implantação, o que fez reduzir o tempo de resposta das requisições de usuários nos cenários envolvendo um elevado número de acessos.

Para testarmos e confirmarmos a aplicabilidade deste trabalho, dedicamos a uma das possíveis formas de como o *framework* pode ser proveitoso para acompanhar e reagir ao aumento no acesso a recursos de um servidor Web visando garantir a estabilidade tanto do sistema quanto do tempo de resposta. Embora reconheçamos a necessidade do avanço de outras pesquisas e de melhorias na implementação do *framework* e de seus componentes, este trabalho representa um passo concreto em busca da construção de um sistema modular capaz de acompanhar cautelosamente a operação dos sistemas sob o seu supervisionamento e readaptar as condições dos ambientes nos quais eles se encontram implantados a fim de melhorar a eficácia das suas atividades.

A característica principal que diferencia o presente *framework* das outras abordagens está na modularidade. Além de melhorar o tempo de resposta e reduzir o número de falhas, que é comum entre as demais abordagens, facilitamos a inserção das políticas de adaptação e não restringimos a escrita de sensores e atuadores em uma linguagem de programação específica. Isso foi possível por meio da exploração de técnicas da engenharia de *software* tais como padrões de projetos e de comunicação interprocessos, disponibilizada pelo sistema

operacional. As peças que compõem o sistema podem ser instaladas, desenvolvidas ou removidas à medida que for necessário.

Um ponto muito importante que exige reflexão, discussão e progresso está relacionado à maneira centralizada de funcionamento do *framework*. Atualmente, ele funciona como uma única instância, o que representa um único ponto de falha, ou seja, se a máquina virtual ou o equipamento no qual ele estiver em execução parar de responder ou for abruptamente desligado, o *framework* não cumprirá seu objetivo. Portanto, ele deve funcionar de maneira distribuída de maneira que uma política de adaptação inserida seja propagada entre outros nós que também contenham suas instâncias em execução.

Durante o desenvolvimento da pesquisa, identificamos outros obstáculos que limitaram o progresso do trabalho. Entre eles, a limitação de tempo foi a mais impactante, porque ela nos impossibilitou de testar diferentes modelos de implantação de sistemas tais como computação em nuvem. Devido ao tempo, também, tivemos que desenvolver uma interface gráfica mais simples a fim de que os sensores, os atuadores e as políticas de adaptação fossem cadastradas. Isso impediu a construção de um sistema muito mais simples de ser gerenciado por qualquer administrador de sistema.

Além dessas limitações que devem ser superadas, o *framework* Adapta deve receber outros avanços impactantes.

O desafio mais importante está relacionado ao aperfeiçoamento das técnicas para inserção das políticas de adaptação. Por exemplo, ao invés de descrever qual propriedade em JSON entre os dados recebidos do receptor deve estar alinhada a determinadas condições específicas, a interface gráfica poderia receber a regra de negócio por meio de fluxogramas usando recursos de “arrastar e soltar”. Outro exemplo aplausível também seria a adoção do modelo de *feature*.

O propósito deste trabalho foi desenvolver uma solução capaz de abordar paliativamente problemas no ambiente de implantação a fim de que os sistemas supervisionados pelo *framework* Adapta continuem em operação até que um administrador trate a causa raiz do problema. Apesar da proposta atualmente desenvolvida abordar as variações nesses ambientes praticamente como estruturas condicionais de linguagem de programação (se o valor coletado pelo sensor satisfizer a uma determinada regra, então aja de determinada maneira, senão verifique se ele atende a outra condição e aja em resposta a ela), o *framework* poderia receber as condições aceitáveis pelo administrador e estar integrado, por exemplo, a técnicas de inteligência artificial visando adequar o ambiente ao que foi estabelecido. É muito importante relembrar que, assim como os sensores e os atuadores são modulares, as políticas de adaptação também são e não há restrição que impeça de integrar o *framework*, futuramente, a diferentes abordagens mais inteligentes para tratamento delas.

Referências

AFFANDI, A. et al. Design and Implementation Fast Response System Monitoring Server Using Simple Network Management Protocol (SNMP). In: IEEE. *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*. [S.l.], 2015. p. 385–390. Citado na página 56.

AL-AYYOUB, M. et al. SDSecurity: A Software Defined Security Experimental Framework. In: IEEE. *Communication Workshop (ICCW), 2015 IEEE International Conference on*. [S.l.], 2015. p. 1871–1876. Citado na página 57.

ALEXANDER, R. T.; BIEMAN, J. M.; ANDREWS, A. A. *Towards the Systematic Testing of Aspect-Oriented Programs*. [S.l.], 2004. Citado na página 58.

ALLIANZ. Allianz Risk Barometer: Results Appendix 2020. *<https://www.agcs.allianz.com/content/dam/onemarketing/agcs/agcs/reports/Allianz-Risk-Barometer-2020-Appendix.pdf>*, 2020. Citado 2 vezes nas páginas 17 e 21.

BENNETT, K. Legacy systems: Coping with success. *IEEE software*, IEEE, v. 12, n. 1, p. 19–23, 1995. Citado 2 vezes nas páginas 18 e 21.

CASE, J. et al. RFC-1157: Simple Network Management Protocol (SNMP). *<https://tools.ietf.org/html/rfc1157>*, RFC Editor, 1990. Citado na página 55.

COUTINHO, E. F. et al. An Architecture for Providing Elasticity Based on Autonomic Computing Concepts. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 412–419. Citado na página 33.

DARABSEH, A. et al. SDDC: A Software Defined Datacenter Experimental Framework. In: IEEE. *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. [S.l.], 2015. p. 189–194. Citado na página 57.

DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs Containerization to support PaaS. In: IEEE. *2014 IEEE International Conference on Cloud Engineering*. [S.l.], 2014. p. 610–614. Citado na página 25.

FILIERI, A. et al. Control Strategies for Self-Adaptive Software Systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, ACM, v. 11, n. 4, p. 24, 2017. Citado na página 40.

FOWLER, M. Microservice trade-offs. *<https://martinfowler.com/articles/microservice-trade-offs.html>*, 2015. Citado na página 17.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing — degrees, models, and applications. *ACM Computing Surveys (CSUR)*, ACM Nova Iorque, NY, Estados Unidos da América, v. 40, n. 3, p. 1–28, 2008. Citado na página 34.

HUNNEBECK, L. *ITIL Service Design 2011 Edition*. 2011. ed. [S.l.]: The Stationery Office, 2011. ISBN 0113313055. Citado 3 vezes nas páginas 18, 22 e 23.

IBM. An architectural blueprint for autonomic computing. *IBM White Paper*, Citeseer, v. 31, p. 1–6, 2006. Citado na página 24.

IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, p. 1–3951, Jan 2018. Citado na página 35.

IRWIN, J. et al. Aspect-oriented programming. *Proceedings of ECOOP, IEEE, Finland*, p. 220–242, 1997. Citado na página 58.

ITSMF. *ITIL Foundation Handbook*. 3ª. ed. GBR: The Stationery Office, 2012. ISBN 0113313497. Citado 2 vezes nas páginas 21 e 22.

KARAKUS, M.; DURRESI, A. A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN). *Computer Networks*, Elsevier, v. 112, p. 279–293, 2017. Citado 2 vezes nas páginas 56 e 57.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE, v. 36, n. 1, p. 41–50, 2003. Citado 2 vezes nas páginas 23 e 24.

LI, H.; CHEN, T.-H.; HASSAN, A. E. Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems: An Industrial Experience Report. 2018. Citado na página 18.

MIRESLAMI, S. et al. Dynamic Cloud Resource Allocation Considering Demand Uncertainty. *IEEE Transactions on Cloud Computing*, IEEE, 2019. Citado na página 33.

MONPERRUS, M. Automatic Software Repair: a Bibliography. *ACM Computing Surveys (CSUR)*, ACM Nova Iorque, NY, Estados Unidos da América, v. 51, n. 1, p. 1–24, 2018. Citado na página 18.

NURSEITOV, N. et al. Comparison of JSON and XML Data Interchange Formats: A Case Study. *Caine*, v. 9, p. 157–162, 2009. Citado na página 40.

PEZOA, F. et al. Foundations of JSON Schema. In: *Proceedings of the 25th International Conference on World Wide Web*. [S.l.: s.n.], 2016. p. 263–273. Citado na página 40.

ROOHI, A.; RAEISIFARD, K.; IBRAHIM, S. An Application for Management and Monitoring the Data Centers Based on SNMP. In: IEEE. *2014 IEEE Student Conference on Research and Development*. [S.l.], 2014. p. 1–4. Citado na página 56.

RUTTEN, E.; MARCHAND, N.; SIMON, D. Feedback Control as MAPE-K Loop in Autonomic Computing. *Software Engineering for Self-Adaptive Systems III. Assurances*, Springer, p. 349–373, 2017. Citado na página 34.

SCHNEIDEWIND, N. F.; EBERT, C. Preserve or redesign legacy systems? *IEEE Software*, IEEE Computer Society, v. 15, n. 4, p. 14, 1998. Citado 2 vezes nas páginas 18 e 21.

TOMOVIC, S.; PRASAD, N.; RADUSINOVIC, I. SDN control framework for QoS provisioning. In: IEEE. *2014 22nd telecommunications forum Telfor (TELFOR)*. [S.l.], 2014. p. 111–114. Citado na página 57.

TURNBULL, J. *The Docker Book: Containerization is the new virtualization*. [S.l.]: James Turnbull, 2014. Citado na página 26.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012. Citado na página 29.

ZHU, M.; PHAM, H. A multi-release software reliability modeling for open source software incorporating dependent fault detection process. *Annals of Operations Research*, Springer, v. 269, n. 1-2, p. 773–790, 2018. Citado na página 17.

Apêndices

APÊNDICE A – Formulário

Formulário

Início

Identificação de Incidentes Frequentes relacionados ao Ambiente Virtual de Aprendizagem

Perfil do(a) Entrevistado(a)

Nome completo *

Cargo/Função *
- Analista de Tecnologia da Informação
- Técnico de Tecnologia da Informação
- Outro(a) cargo ou função

Qual cargo ou função?

Instituição *

O Moodle ou outro Ambiente Virtual de Aprendizagem (AVA) que você administra tem muitos usuários? Qual é o número aproximado? *

Tempo de Experiência

Implantação e Administração de Sistemas * — nenhuma experiência

Banco de Dados * — nenhuma experiência

Programação * — nenhuma experiência

Moodle ou outro AVA * — nenhuma experiência

Questionário

Qual tipo de configuração você tem usado para instalar o Moodle ou outro AVA no seu ambiente? *

- ○ O Servidor Web e o SGBD têm sido instalados no mesmo servidor
- ○ O Servidor Web e o SGBD têm sido instalados em servidores diferentes
- ○ Tem-se usado um servidor de hospedagem ou uma solução em nuvem (Cloud), e eu não sei como o servidor Web e o SGBD estão instalados
- ◉ Tem sido usada uma configuração diferente (por favor, descreva-a no espaço que aparecerá a seguir)

Descreva, por gentileza, a sua configuração

Na prática, qual(is) incidente(s) você observa frequentemente (pode marcar mais de uma opção)? Quanto tempo você leva para corrigi-lo(s)?

- ☑ problema de conexão com o SGBD (devido à sobrecarga no SGBD)

Quanto tempo você leva para corrigi-lo?

menos de 30 minutos

- ☑ sobrecarga no servidor Web

Quanto tempo você leva para corrigi-lo?

menos de 30 minutos

- ☑ vulnerabilidades no Ambiente Virtual de Aprendizagem (AVA) ou nos seus módulos

Quanto tempo você leva para corrigi-lo?

menos de 30 minutos

- ☑ outro(s)

Quais são e quanto tempo você leva para corrigi-los?

Quando um incidente acontece (perda de conexão, SGBD inacessível, o servidor Web para de responder e assim por diante), como você identifica que ele aconteceu? Alguém te avisa, você inspeciona periodicamente o sistema ou há outro sistema que te alerta automaticamente? *

Quando um incidente é identificado, quem normalmente resolve o problema tem uma longa experiência no Moodle ou em outro AVA usado pela sua instituição? Quanto tempo de experiência? Esse profissional tem conhecimento avançado em programação? *

Você tem usado arquivos do Moodle ou de outro AVA compartilhado entre servidores, máquinas virtuais ou contêineres (e.g. NFS, SMB, FTP, volumes compartilhados entre vários contêineres do Docker etc)? Se você tem, quais dessas ou outras tecnologias você usa? *

As funções do AVA podem ser estendidas com o uso de módulos após a sua instalação. Na sua opinião, quais módulos são mais suscetíveis a incidentes no ambiente onde você os usa (por exemplo: questionário, fórum, bate-papo etc)? Considere como alguns incidentes os exemplos seguintes: perda de conexão, sobrecarga (negação de serviço), consumo excessivo de espaço em disco, SGBD inacessível etc. *

Enviar