Uma Estratégia Orientada a Aspectos para Monitoramento de Plataformas para Cidades Inteligentes

André Solino, João Victor Lopes, Thais Batista, Everton Cavalcante, Jorge Pereira, Aluízio Rocha Neto

Universidade Federal do Rio Grande do Norte (UFRN)
Natal-RN, Brasil

{andresolino, aluizio}@imd.ufrn.br, joao.victor_lopes27@outlook.com
{jorgepereirasb, thaisbatista}@gmail.com, everton.cavalcante@ufrn.br

Resumo. *Plataformas para cidades inteligentes tipicamente oferecem importantes funcionalidades que visam facilitar o desenvolvimento de aplicações. Uma das características desse cenário está relacionada ao alto volume de requisições e de dados tratados, fazendo com que seja necessário monitorar a infraestrutura computacional subjacente sobre a qual plataformas para cidades inteligentes e as aplicações desenvolvidas estão implantadas para que elas sejam mais escaláveis e mantenham sua qualidade de serviço. Este artigo apresenta uma estratégia não invasiva para possibilitar o monitoramento de plataformas para cidades inteligentes. A estratégia proposta apoia-se no paradigma de programação orientada a aspectos para que seja possível monitorar a infraestrutura computacional sem a necessidade de intervir sobre a implementação da plataforma ou gerar acoplamento com relação ao monitoramento. Este trabalho apresenta ainda a implementação da estratégia e sua instanciação no monitoramento na plataforma Smart Geo Layers (SGeoL).*

Abstract. *Smart city platforms typically offer important functionalities to ease application development. One of the characteristics of this scenario is related to the high volume of requests and the handled data, requiring monitoring of the underlying computing infrastructure on which smart city platforms and the developed applications are deployed to make them more scalable and keep their quality of service. This paper presents a non-invasive strategy to monitor smart city platforms. The proposed strategy relies on the aspect-oriented programming paradigm to monitor the computational infrastructure without intervening in the platform's implementation or generating coupling with the monitoring capabilities. This work also presents the implementation of the strategy and its instantiation for monitoring the Smart Geo Layers (SGeoL) platform.*

1. Introdução

Plataformas para cidades inteligentes têm como principal objetivo facilitar o desenvolvimento, execução e implantação de aplicações nesse contexto. Tais plataformas tipicamente oferecem importantes funcionalidades na forma de serviços reutilizáveis que os desenvolvedores de aplicações podem usar, por exemplo, o gerenciamento e armazenamento de grandes volumes de dados, coleta e análise de dados, integração de dispositivos, serviços e sistemas heterogêneos, dentre outros [Santana et al. 2017]. Alguns requisitos de qualidade também são relevantes no escopo dessas plataformas, tais como interoperabilidade, privacidade, adaptabilidade e escalabilidade.

No que diz respeito à escalabilidade, uma plataforma para cidades inteligentes deve ser capaz de lidar com um grande e crescente número de usuários, dispositivos, serviços e aplicações envolvidos. A plataforma necessita armazenar e processar grandes volumes de dados relacionados à cidade, os quais são continuamente produzidos e consumidos por dispositivos e aplicações de forma simultânea, além de dar suporte a milhares de requisições de usuários e das aplicações que estejam fazendo uso de seus serviços. A demanda requisitada à plataforma pode ser bastante variável de acordo com as características das aplicações, dos usuários e da própria cidade como um todo [Del Esposte et al. 2019].

Diversas plataformas já foram propostas na literatura para endereçar desafios relacionados ao desenvolvimento e à implantação de aplicações e serviços para cidades inteligentes [Santana et al. 2017], todavia, a escalabilidade dessas soluções não tem sido o foco da pesquisa nesse contexto [Del Esposte et al. 2019]. Uma plataforma para cidades inteligentes pode sofrer sobrecargas (inclusive imprevistas) durante a sua operação, de modo que tais situações devem ser identificadas por meio do monitoramento tanto da operação da plataforma quanto da infraestrutura computacional a ela subjacente. Através desse monitoramento, é possível evitar uma degradação de desempenho ou mesmo interrupção dos serviços da plataforma, além de contribuir com a sua escalabilidade em termos de assimilar quantidades significativas de requisições, dispositivos e usuários.

O monitoramento da operação de uma plataforma para cidades inteligentes deve ser automatizado para possibilitar uma rápida identificação de gargalos. Isso poderá desencadear a realização de ações dinamicamente, idealmente também de forma automatizada, com vistas a evitar degradação de desempenho ou interrupção de serviços, a exemplo de uma (re)alocação de recursos na infraestrutura computacional subjacente para melhor acomodar a demanda corrente à plataforma. As estratégias de monitoramento muitas vezes envolvem algum tipo de instrumentação do código fonte, em que a implementação da plataforma necessita ser modificada para dar suporte ao monitoramento. No caso de plataformas de terceiros, essa abordagem mais invasiva requer analisar a implementação da plataforma (isto se ela for de código aberto) e incluir as funcionalidades associadas ao monitoramento, o que pode envolver um esforço significativo de desenvolvimento.

O objetivo deste trabalho é apresentar uma estratégia não invasiva para possibilitar o monitoramento de plataformas para cidades inteligentes. A estratégia proposta apoia-se no paradigma de programação orientada a aspectos [Kiczales et al. 1997] para que seja possível monitorar a operação da plataforma e o uso dos recursos da infraestrutura computacional a ela subjacente, sem a necessidade de intervir sobre a sua implementação ou gerar acoplamento com relação ao monitoramento. O monitoramento ocorre em diferentes níveis (*host*, *container*, plataforma) para obter uma visão geral do comportamento da infraestrutura e da plataforma implantada sobre esta. Neste trabalho, a estratégia em questão foi implementada e instanciada para o monitoramento da *Smart Geo Layers* (SGeoL) [Pereira et al. 2022], uma plataforma georreferenciada para cidades inteligentes que integra dados heterogêneos de diferentes domínios de uma cidade.

Este artigo está estruturado da seguinte forma. A Seção 2 apresenta conceitos básicos relacionados a este trabalho. A Seção 3 apresenta a estratégia de monitoramento orientada a aspectos proposta. A Seção 4 descreve a implementação da estratégia de monitoramento e sua instanciação para a plataforma SGeoL. A Seção 5 discute algumas propostas relacionadas apresentadas na literatura. A Seção 6 traz considerações finais.

2. Conceitos Básicos

2.1. Monitoramento em Cidades Inteligentes

Uma plataforma para cidades inteligentes funciona como um *middleware* que provê serviços distribuídos comuns a diversas aplicações e usuários e agrega bases de dados para o armazenamento, processamento, análise e visualização dos dados por ela gerenciados. Plataformas para cidades inteligentes são tipicamente implantadas em infraestruturas computacionais baseadas em nuvem, com vistas a garantir maior escalabilidade e disponibilidade. Portanto, os elementos relevantes para o monitoramento nesse contexto são (i) as operações internas da plataforma, (ii) as consultas realizadas sobre os bancos de dados por ela gerenciados e (iii) os servidores sobre os quais ela está implantada.

As funcionalidades disponibilizadas por uma plataforma para cidades inteligentes em geral são expostas através de métodos acessíveis por meio de APIs. Como a plataforma pode ser utilizada por múltiplas aplicações simultaneamente, é importante monitorar os métodos invocados para que seja possível identificar qual deles esteja causando uma eventual degradação de desempenho. Nesse caso, a métrica que pode ser coletada mediante o monitoramento da plataforma é o **tempo de resposta** desses métodos para finalizar a operação requisitada. É possível ainda coletar a quantidade de solicitações que são atendidas por ela em um determinado período de tempo (***throughput***). Essas métricas em conjunto possibilitam assim ter uma visão abrangente acerca do comportamento da plataforma.

A operação de uma plataforma para cidades inteligentes frequentemente envolve várias consultas a bancos de dados, as quais podem inclusive ser custosas em termos de processamento e necessitam lidar com grandes volumes de dados. Além das operações internas, é relevante monitorar as consultas realizadas sobre os bancos de dados gerenciados pela plataforma. Por exemplo, uma operação de consulta envolvendo um volume significativo de dados geográficos pode requerer bastante tempo para ser executada caso o banco de dados esteja em uma máquina com recursos (CPU, memória) insuficientes para realizar tal operação. Dessa forma, é importante monitorar os bancos de dados para verificar se há alguma limitação, uma vez que eles precisam ter recursos computacionais suficientes para processarem grandes quantidade de dados em um tempo aceitável. De acordo com [Araujo et al. 2019], a métrica relevante para monitoramento sobre os bancos de dados responsáveis por armazenarem dados gerados pelas diversas aplicações e dispositivos é a **taxa de transferência** de dados.

Plataformas para cidades inteligentes e seus bancos de dados costumam ser implantados em servidores disponíveis em um ambiente de Computação em Nuvem, incluindo máquinas virtuais e *containers*. Assim sendo, pode ser necessário monitorar uma quantidade significativa de *containers* e de máquinas virtuais dependendo da quantidade de componentes integrados à plataforma. É importante nesse caso monitorar métricas como **uso de CPU**, **memória**, **disco** e **rede** tanto das máquinas virtuais quanto dos contêineres que executam sobre o servidor. O monitoramento dos diferentes níveis de virtualização permite assim ter uma visão geral do comportamento da infraestrutura computacional que dá suporte à implantação e execução de uma plataforma para cidades inteligentes.

2.2. Programação Orientada a Aspectos

A programação orientada a aspectos (POA) [Kiczales et al. 1997] propõe uma solução para a modularização e composição de interesses que são transversais a várias partes de um sistema, os chamados *crosscutting concerns*. Um interesse transversal refere-se a uma

determinada parte de um programa que afeta muitas outras partes dele. Em razão disso, a sua implementação apresenta-se dispersa em diversos componentes responsáveis pelas funcionalidades básicas do sistema (código base), violando princípios básicos do projeto de *software*. São exemplos comuns de interesses transversais *logging*, autenticação, tratamento de exceções, persistência, entre outros. A POA possibilita que as funcionalidades básicas de uma aplicação estejam separadas dos interesses transversais, aumentando sua reusabilidade e facilitando sua manutenção e legibilidade.

As abstrações da POA para a representação dos interesses transversais são *aspectos*, *join points*, *pointcuts* e *advices*. Os *aspectos* implementam e encapsulam interesses transversais por meio de instruções sobre onde, quando e como eles são invocados. Dessa forma, o aspecto promete uma maior modularidade ao evitar misturar o código das funcionalidades do sistema com o código dos interesses transversais. *Join points* são locais bem definidos na estrutura ou fluxo de execução de um programa onde comportamentos definidos pelos aspectos podem ser inseridos. *Pointcuts* reúnem um conjunto de *join points* que são afetados por um ou mais interesses transversais.

Um *advice* é um conjunto de operações que são executadas quando os *join points* especificados são alcançados. Normalmente, cada *advice* possui um *pointcut* a ele associado, o qual determina os *join points* onde esse *advice será executado*. As declarações de *advice* podem ser de três tipos: (i) *before*, para executar o *advice* no momento em que um *join point* é alcançado; (ii) *after*, para executar o *advice* no momento em que o controle retorna através do *join point*, e; (iii) *around*, para executar o *advice* quando o *pointcut* é alcançado e tem controle explícito sobre quando o método afetado deve ser executado.

Enquanto o código base da aplicação é implementado usando uma linguagem de programação tradicional, os aspectos são implementados usando uma linguagem orientada a aspectos como AspectJ[1] [Kiczales et al. 2001], uma extensão da linguagem de programação Java para POA. Um processo de *weaving* une o código definido pelo aspecto com o código base, injetando o *advice* nos *join points* definidos. A Figura 1 ilustra o processo de *weaving* na POA. Um aspecto é implementado separadamente do código base da aplicação que será afetado pelos *pointcuts* definidos no aspecto. Os *pointcuts* são definidos para interceptar um conjunto de *join points* no código base, sendo a interação entre ele e o aspecto definida através do *advice*. Para realizar o processo de composição, o *weaver* identifica quais *join points* serão interceptados pelos *pointcuts* definidos e produz um código composto resultante da inserção do código contido no *advice* definido pelo aspecto no código base da aplicação.

Dado que o código base e o aspecto são definidos separadamente e que o código base não precisa ter conhecimento do código do aspecto, a POA é bastante apropriada para situações em que é necessário inserir um determinado interesse transversal na aplicação, mas deseja-se evitar o uso de uma abordagem invasiva que modifique o código base. Essas características motivaram a escolha da POA para concretização da estratégia de monitoramento de plataformas para cidades inteligentes proposta neste trabalho.

3. Estratégia de Monitoramento

A estratégia de monitoramento proposta neste trabalho realiza o monitoramento em múltiplos níveis, os quais são ilustrados na Figura 2. Com relação à infraestrutura subjacente

[1]https://www.eclipse.org/aspectj/

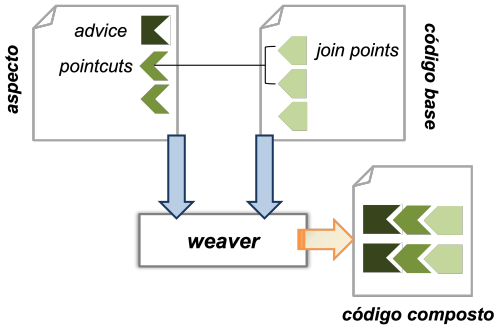

Figura 1. Abstrações da POA e o processo de *weaving*

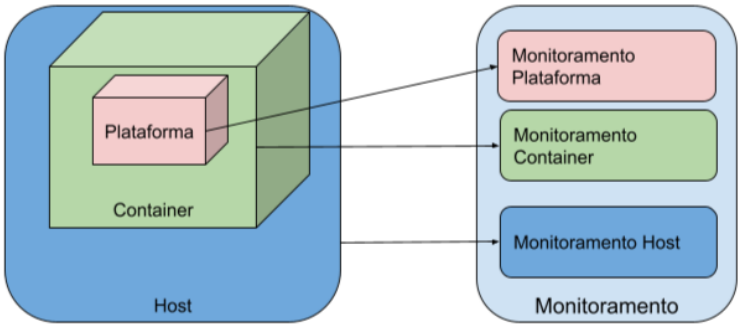

Figura 2. Níveis de monitoramento da estratégia proposta

que dá suporte à plataforma, são monitoradas métricas em dois níveis, do *host*[2] e de *containers*, caso estes sejam utilizados. Nesses níveis, são coletadas métricas de consumo de CPU, interface de rede e memória pelo *host* e pelos *containers*. Por sua vez, o monitoramento a nível da plataforma coleta métricas como o tempo despendido na execução de uma determinada operação interna da plataforma, por exemplo, consultas para acesso a dados por ela gerenciados.

Há várias abordagens e ferramentas (inclusive de código aberto) disponíveis para concretizar os níveis de monitoramento de *host* e de *containers*, tais como Elasticsearch[3], Prometheus[4] e NetData[5]. O monitoramento em tais níveis é comumente realizado por meio da implantação de agentes de monitoramento responsáveis pela coleta das métricas desejadas. Esses agentes podem enviar as métricas monitoradas para um servidor de banco de dados, um serviço de mensagens ou mesmo algum outro serviço de monitoramento. As soluções de monitoramento disponíveis em alguns estudos na literatura consideram métricas como CPU, memória, rede, requisições HTTP, dentre outras [Abranches and Solis 2016, Matsumoto et al. 2019, Narayana et al. 2020]. Esses trabalhos monitoram os níveis de *host* e de *container*, porém em nenhum deles o monitoramento foi realizado considerando métricas relativas à operação interna da plataforma, o que se constitui um diferencial deste trabalho frente ao estado da arte.

Utilizar um agente de monitoramento não é suficiente para o monitoramento da estrutura interna da plataforma, pois isso requer o conhecimento de tal estrutura. O agente

[2]No contexto deste trabalho, o termo *host* é utilizado para identificar tanto uma máquina virtual (VM) quanto uma máquina física (*bare metal*).
[3]https://www.elastic.co/elasticsearch/
[4]https://prometheus.io/
[5]https://www.netdata.cloud/agent/

de monitoramento possui a visão externa da plataforma, como um componente opaco, não possuindo acesso a detalhes internos. O tempo despendido na execução de uma determinada operação da plataforma, por exemplo, pode ser o somatório dos tempos de execução de outras operações chamadas pela primeira. Com o monitoramento da estrutura interna, é possível saber o tempo despendido em cada uma dessas operações individualmente e identificar qual delas está causando eventual degradação do desempenho.

3.1. Monitoramento Utilizando POA

Para obter as informações necessárias ao monitoramento interno da plataforma, normalmente é necessário alterar a sua implementação para inserir código fonte responsável por coletar essas informações. Entretanto, alterar o código fonte da plataforma alvo do monitoramento é uma abordagem invasiva e que traz complexidade para se efetivar o monitoramento, sendo necessário conhecer de forma detalhada a implementação da plataforma.

A estratégia de monitoramento proposta neste trabalho apoia-se no uso de POA para a coleta dos dados necessários ao monitoramento da plataforma sem a necessidade de conhecer o código base de sua implementação nem alterá-lo incluindo diretivas de monitoramento. Como uma estratégia modular, é necessário apenas conhecer as interfaces das operações providas pela plataforma, de modo que o aspecto associado ao monitoramento pode definir os *join points* para cada operação e os respectivos *advices* para antes e depois da chamada à operação monitorada. Através do processo de *weaving*, o código fonte da plataforma é utilizado em conjunto com as definições de aspectos criados para o monitoramento. A Figura 3 ilustra a transversalidade entre uma plataforma e um aspecto de monitoramento que define *join points* no código base sobre o qual ele irá atuar.

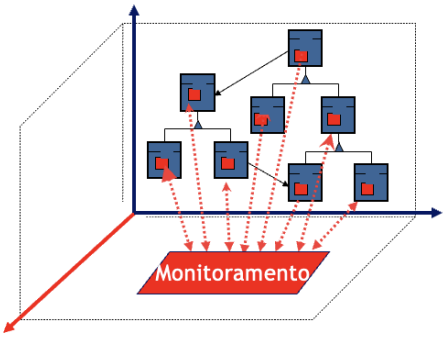

Figura 3. Aspecto de monitoramento e *join points*

Na estratégia de monitoramento proposta neste trabalho, as operações críticas da plataforma são monitoradas por meio de *join points*, havendo um *join point* associado a cada uma dessas operações toda vez que ela é invocada. Para cada *join point* há um *advice* responsável por coletar as informações de monitoramento desejadas, como, por exemplo, o tempo despendido na execução de cada operação individualmente e, por consequência, o tempo total da execução de uma operação que envolva a chamada de outras operações.

3.2. Arquitetura de Monitoramento

A Figura 4 apresenta uma arquitetura para a concretização da estratégia de monitoramento proposta neste trabalho. Para cada elemento alvo do monitoramento, seja ele o *host*, *container* ou a plataforma, há um *agente de monitoramento* responsável pela coleta

das métricas. Agentes de monitoramento podem inclusive coletar dados de mais de um nível de monitoramento.

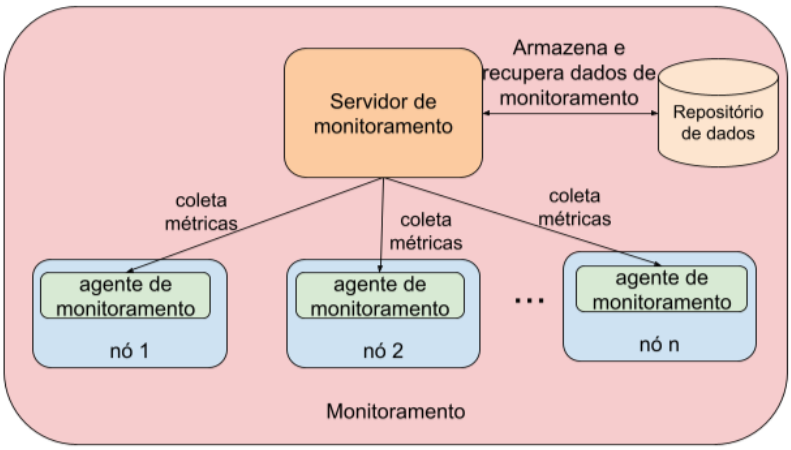

Figura 4. Arquitetura proposta para o monitoramento

Os agentes de monitoramento coletam as métricas, as quais são centralizadas em um servidor de monitoramento para subsequente armazenamento em um repositório de dados. A coleta das métricas pode ocorrer de duas formas: na primeira, o agente é responsável por enviar as métricas para o servidor de monitoramento, enquanto na segunda o servidor de monitoramento realiza periodicamente consultas aos agentes para obter os dados. Através do servidor de monitoramento, é possível analisar os dados coletados e avaliar o desempenho da plataforma como um todo, visto que são armazenados os dados de monitoramento de componentes de infraestrutura e da própria plataforma.

4. Implementação e Validação

A estratégia de monitoramento orientada a aspectos proposta neste trabalho foi implementada sobre a SGeoL [Pereira et al. 2022], uma plataforma escalável para o desenvolvimento de aplicações para cidades inteligentes. A plataforma SGeoL oferece diversas funcionalidades que facilitam o desenvolvimento de aplicações, tais como gerenciamento de dados geográficos, informações de contexto e séries temporais de dados, além da integração de dados heterogêneos e visualização de informações geográficas e analíticas. Todas essas funcionalidades são providas por APIs e componentes organizados de maneira distribuída, e que colaboram entre si para prover as funcionalidades supracitadas. Atualmente, a plataforma SGeoL executa sobre um *cluster* composto por três máquinas virtuais, cada uma delas contando com seis VCPUs 2.6 GHz, 6 GB de memória RAM e sistema operacional Ubuntu Server versão 20.04. Sobre essas máquinas, foram instanciados oito *containers* contendo os serviços que fazem parte da plataforma.

A plataforma SGeoL lida com um grande volume de informações, principalmente dados geográficos, além de executar tarefas complexas e custosas, tais como consultas geográficas e importação de dados heterogêneos disponíveis através de arquivos, planilhas ou APIs de outros sistemas. Portanto, é importante monitorar as ações da plataforma bem como as requisições dos usuários a fim de identificar possíveis problemas de desempenho na plataforma. Com base no monitoramento, seria possível disparar processos de escalonamento automático (*autoscaling*) a fim de evitar sobrecargas na infraestrutura da

plataforma, oferecendo assim uma boa experiência de uso de seus serviços e evitar sub ou superprovisionamento de recursos computacionais na infraestrutura.

Para implementar a arquitetura proposta para o monitoramento da infraestrutura subjacente sobre a qual a plataforma SGeoL está implantada e em execução (*host e container*), foi utilizado um conjunto composto das seguintes ferramentas, todas de código aberto: Elasticsearch[6], Metricbeat[7], Filebeat[8] e Kibana[9]. A escolha por tais ferramentas deveu-se ao fato de elas oferecerem as funcionalidades necessárias à implementação da estratégia de monitoramento, em termos das métricas coletadas e armazenadas, além de serem relativamente fáceis de configurar e serem integradas umas às outras [Bagnasco et al. 2015]. O Metricbeat é o agente de monitoramento responsável por coletar as métricas no nível de *host* e *container*, enquanto o Filebeats coleta métricas no nível da plataforma. Ambos os dados coletados são enviados para o mesmo banco de dados, possibilitando a integração dos dados de monitoramento dos vários níveis e assim uma visão holística do monitoramento da plataforma.

Com relação ao monitoramento das operações da plataforma, para recuperar as informações necessárias, o código fonte da plataforma SGeoL, originalmente na linguagem de programação Java, foi instrumentado com a utilização da POA. Para fins de exemplo, o Código 1 apresenta a implementação do aspecto *LayerResourceAspect* na linguagem orientada a aspectos AspectJ. Na linha 5 é criado um *pointcut allMethods()* para todos os métodos da classe *LayerResource*. Na linha 8 é criado um *advice* do tipo *around* para esse *pointcut allMethods*. Todas as vezes que algum método da classe *LayerResource* for executado, o código representado pelo aspecto será invocado e exibirá na saída padrão o método interceptado e o tempo para sua execução. Note-se, portanto, que não foi necessário realizar qualquer alteração na implementação da classe *LayerResource* presente no código fonte da plataforma SGeoL, evidenciando assim o benefício da adoção do paradigma de POA para o possibilitar o monitoramento.

A Figura 5 apresenta a implementação do monitoramento da plataforma SGeoL. O código da plataforma foi instrumentado como resultado do processo de *weaving* (ver Seção 2.2) do aspecto implementado para o monitoramento e o código base, gerando assim as informações de monitoramento desejadas. Essas informações são exibidas na saída padrão e são capturadas pelo Filebeat, que as envia para o Elasticsearch. O Metricbeat coleta as métricas do *container* e do *host*. No caso específico da plataforma SGeoL, o Metricbeat é executado em um *container*, possibilitando o monitoramento em nível de *container*, e os dados resultantes são enviados para o Elasticsearch. Finalmente, o Kibana é utilizado para analisar os dados armazenados no Elasticsearch.

[6]https://www.elastic.co/elasticsearch/
[7]https://www.elastic.co/beats/metricbeat
[8]https://www.elastic.co/beats/filebeat
[9]https://www.elastic.co/pt/kibana/

```
1  package sgeol.sintactic_api.resouces;
2
3  public aspect LayerResourceAspect {
4      // Pointcut para todos os metodos da classe LayerResource
5      pointcut allMethods() : execution(public * LayerResource.*(..));
6
7      // Advice para calcular o tempo de execucao do metodo
8      Object around(): allMethods(){
9          Long beginTime = System.currentTimeMillis();
10         Object object = proceed();
11         Long endTime = System.currentTimeMillis();
12         System.out.println("[monitoramento] "  + thisJoinPoint.
               getSignature() + " " + (endTime - beginTime));
13         return object;
14     }
15 }
```

Código 1. Implementação em AspectJ do aspecto *LayerResourceAspect* para monitoramento na plataforma SGeoL

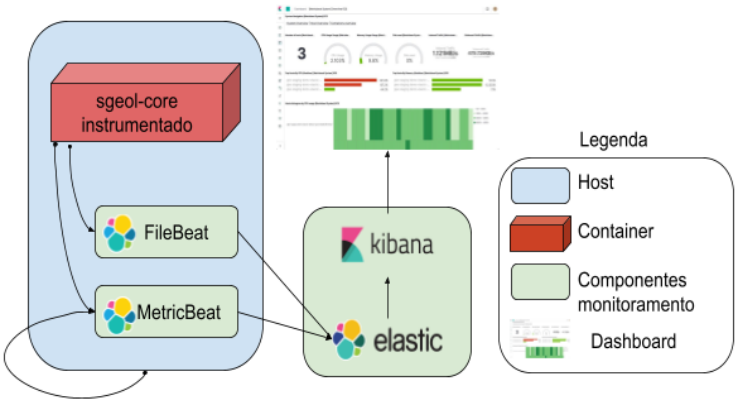

Figura 5. Implementação da arquitetura proposta

A implementação realizada possibilitou validar (i) que a estratégia proposta é capaz de realizar o monitoramento de forma não invasiva, e (ii) que foi possível obter métricas das operações internas da plataforma que normalmente não são acessíveis a agentes de monitoramento. O monitoramento foi realizado sem alterar o código fonte da plataforma SGeoL, em função da adoção do paradigma de POA. A inserção do aspecto apresentado no Código 1 permitiu a captura das métricas de monitoramento de qualquer método da classe *LayerResource*. Com isso, concluiu-se que o uso de POA foi relevante para a implementação da arquitetura proposta, tendo em vista a complexidade do monitoramento e manutenção de plataformas para cidades inteligentes, como é o caso da SGeoL.

A estratégia proposta permitiu ainda identificar as operações mais onerosas em termos de tempo de resposta na plataforma SGeoL. A SGeoL disponibiliza para seus usuários dezenas de operações por meio de sua API RESTful, as quais foram monitoradas utilizando a implementação do aspecto para contabilizar o tempo despendido pela plataforma para processa-las quando acessadas pelos usuários. Dessa forma, foi possível notar através do monitoramento que, dentre as operações disponibilizadas, as operações

que lidam diretamente com processamento geográfico e importação de dados via arquivos vetoriais do tipo *shapefile* são as que demandam mais tempo para serem atendidas. No que diz respeito às operações que demandam processamento geográfico, observou-se ainda que maior parte do tempo despendido nessas operações é consumida pelas chamadas ao banco que armazena dados geográficos. Esses dados permitiram redimensionar a infraestrutura que suporta esse banco de dados a fim de obter melhores tempos de resposta especificamente para essas operações.

5. Trabalhos Relacionados

[Araujo et al. 2019] realizaram uma avaliação de desempenho da plataforma europeia FIWARE[10] utilizando como métricas uso de CPU, memória e taxa de transferência, sobre uma infraestrutura de monitoramento baseada no Prometheus. As métricas obtidas por meio de ferramentas de monitoramento e instrumentação do código fonte possibilitaram a visualização do comportamento dos componentes da plataforma, mais especificamente agentes de Internet das Coisas, o componente de gerenciamento de contexto e o banco de dados da plataforma. Nesse trabalho, a instrumentação do código fonte para o monitoramento representa uma abordagem invasiva, pois as bibliotecas de clientes disponibilizadas pelo Prometheus exigem implementação dentro do código da aplicação. Ainda assim, a avaliação realizada conseguiu aferir informações acerca do desempenho da plataforma e visualizar gargalos, concluindo que a limitação da plataforma encontrava-se no banco de dados responsável por armazenar os dados provenientes dos diferentes dispositivos de Internet das Coisas.

[Del Esposte et al. 2019] apresentam uma avaliação da plataforma InterSCity[11] por meio de simulações em larga escala. O monitoramento foi realizado por meio da própria plataforma de orquestração de *containers* Kubernetes, tendo como objeto do monitoramento os *containers* dos microsserviços que implementam as funcionalidades da plataforma. O mecanismo de monitoramento do Kubernetes considera o uso de CPU dos *containers* para analisar o comportamento da plataforma. Contudo, o monitoramento baseado apenas nessa métrica não demonstra de maneira abrangente o comportamento da plataforma, de modo que outras métricas deveriam ser utilizadas para se obter uma visão mais ampla da operação dos componentes de uma plataforma para cidades inteligentes.

[Casalicchio 2019] realizou um estudo sobre medidas de desempenho para monitorar os *containers* de aplicações, considerando o uso de CPU destes juntamente com o uso de CPU na máquina responsável pela sua execução. O estudo foi realizado sobre uma infraestrutura de monitoramento composta de cAdvisor, Prometheus e Grafana. O uso de métricas de monitoramento dos *containers* e das máquinas virtuais permite uma visualização do comportamento da plataforma nesses dois níveis. Todavia, esse trabalho não levou em consideração métricas no nível das aplicações que executam dentro dos *containers*, não sendo possível visualizar o comportamento interno das operações da aplicação.

[Taherizadeh and Stankovski 2019] fazem uso de uma abordagem de monitoramento em múltiplos níveis, a saber de *containers* e de aplicação. O monitoramento no nível de *containers* considera como métricas o uso de CPU, memória, rede e disco, enquanto o monitoramento no nível de aplicação considera o tempo de resposta a requisições e o *throughput*. A abordagem de monitoramento proposta nesse trabalho envolve

[10]https://www.fiware.org

[11]https://interscity.org/software/interscity-platform/

dois elementos, agente e gerente de monitoramento, os agentes sendo implantados dentro dos *containers* juntamente com a aplicação para a coleta das métricas de monitoramento. Os resultados apresentados pelos autores mostraram que o monitoramento atestou que os recursos provisionados não excederam a capacidade exigida e nem foram escassos aos propósitos da aplicação. Entretanto, esse trabalho realiza uma abordagem de monitoramento invasiva utilizando o protocolo *StadsD*[12], em que foi necessário implementar na aplicação os elementos necessários ao monitoramento para transmitir os dados coletados da aplicação para o agente de monitoramento responsável pela coleta.

6. Conclusão

Este artigo apresentou uma estratégia orientada a aspectos para monitoramento de plataformas de cidades inteligentes. Tal estratégia realiza monitoramento em vários níveis, desde a infraestrutura computacional (*host* e *containers*) até as operações realizadas pela plataforma. Diversas métricas são consideradas no monitoramento de acordo com os respectivos níveis, incluindo o consumo de CPU, memória e interface de rede do *host* e de *containers* e o tempo despendido na execução de operações internas da plataforma. O monitoramento nesses três níveis constitui-se como uma das principais contribuições deste trabalho, considerando que as propostas encontradas na literatura frequentemente consideram apenas os níveis de *host* e de *container*, e não a operação interna da plataforma. Outra contribuição deste trabalho é o uso de POA para realizar um monitoramento não invasivo de plataformas para cidades inteligentes, fazendo com que tal monitoramento seja modular, desacoplado e não requeira alterações na implementação da plataforma.

Como prova de conceito, a estratégia proposta foi aplicada na plataforma SGeoL, onde foram coletadas métricas a respeito do tempo de resposta das requisições feitas pelos usuários da plataforma à sua API. O monitoramento permitiu identificar quais as operações demandam mais tempo de processamento para serem atendidas, dando a oportunidade para os desenvolvedores e administradores da plataforma intervirem diretamente sobre essas operações e sobre a infraestrutura subjacente da plataforma a fim de melhorar o seu desempenho. Cabe salientar que, apesar de a estratégia ter sido validada no contexto da plataforma SGeoL, ela poderia ser também concretizada para outras plataformas para cidades inteligentes, aproveitando as facilidades do uso de POA para monitoramento da infraestrutura computacional e das operações da plataforma.

Em termos de trabalhos em andamento e futuros, a estratégia de monitoramento descrita neste artigo vem sendo integrada a uma abordagem baseada no *loop* de controle IBM MAPE-K [IBM 2003] no intuito de prover, de forma autonômica, elasticidade a plataformas para cidades inteligentes. O MAPE-K é um modelo de referência para o desenvolvimento de sistemas autonômicos em diversos contextos que inclui etapas de monitoramento, análise, planejamento e execução de ações, as quais possibilitam que um sistema consiga se autogerenciar em tempo de execução. Com isso, será possível a infraestrutura computacional subjacente que dá suporte à implantação e execução da plataforma ser ajustada dinamicamente de acordo com a carga de trabalho, com vistas a manter os níveis de qualidade de serviço desejados.

[12]https://github.com/b/statsd_spec

Referências

Abranches, M. C. Solis, P. (2016). A mechanism of auto elasticity based on response times for cloud computer enviroments and autossimilar workload. In *Proceedings of the XLII Latin American Computing Conference*, USA. IEEE.

Araujo, V., Mitra, K., Saguna, S., Åohlund, C. (2019). Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities. *Journal of Parallel and Distributed Computing*, 132:250–261.

Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M., Vallero, S. (2015). Monitoring of IaaS and scientific applications on the cloud using the elasticsearch ecosystem. *Journal of Physics: Conference Series*, 608:012016.

Casalicchio, E. (2019). A study on performance measures for auto-scaling CPU-intensive containerized applications. *Cluster Computing*, 22(3):995–1006.

Del Esposte, A. M. et al. (2019). Design and evaluation of a scalable smart city software platform with large-scale simulations. *Future Generation Computer Systems*, 93:427–441.

IBM (2003). An architectural blueprint for Autonomic Computing. Technical report, IBM.

Kiczales, G. et al. (1997). Aspect-oriented programming. In Akşit, M. Matsuoka, S., editores, *ECOOP'97 – Object Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, páginas 220–242. Springer Berlin Heidelberg, Germany.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G. (2001). An overview of AspectJ. In Knudsen, J. L., editor, *ECOOP 2001 – Object-Oriented Programming*, volume 2072 of *Lecture Notes in Computer Science*, páginas 327–354. Springer-Verlag Berlin Heidelberg, Germany.

Matsumoto, R., Kondo, U., Kuribayashi, K. (2019). FastContainer: A homeostatic system architecture high-speed adapting execution environment changes. In *Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference*, páginas 270–275, USA. IEEE.

Narayana, S., Mainak, S., Paul, A. S. (2020). Application deployment using containers with auto-scaling for microservices in cloud environment. *Journal of Network and Computer Applications*, 160.

Pereira, J., Batista, T., Cavalcante, E., Souza, A., Lopes, F., Cacho, N. (2022). A platform for integrating heterogeneous data and developing smart city applications. *Future Generation Computer Systems*, 128:552–566.

Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., Milojičić, D. S. (2017). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys*, 50(6).

Taherizadeh, S. Stankovski, V. (2019). Dynamic multi-level auto-scaling rules for containerized applications. *The Computer Journal*, 62(2):174–197.