Um relato sobre a migração de uma plataforma de *offloading* para microsserviços

Adriano L. Cândido[1,4], Fernando A. M. Trinta[1], Paulo A. L. Rego [1], Lincoln S Rocha[1], Nabor C. Mendonça [2], Vinicius C. Garcia[3]

[1]Departamento de Ciências da Computação da Universidade Federal do Ceará (UFC)
Campus do Pici, s/n, CEP 60451-970 Fortaleza, Ceará, Brasil

[2]Centro de Ciências Tecnológicas - Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, CEP 60811-905 Fortaleza, Ceará, Brasil

[3]Universidade Federal de Pernambuco (UFPE)
Av. Prof. Moraes Rego, 1235, CEP 50670-901 Recife, Pernambuco, Brasil

[4]Faculdade Vale do Salgado (FVS)
Rua Monsenhor Frota, 609, CEP 63430-000 Icó, Ceará, Brasil

Abstract. *Mobile devices are increasingly present in people's daily lives. However, despite the evolution of new generations of smartphones, the amount of information and the complexity of the procedures delegated to these devices still impose processing restrictions, mainly related to energy consumption. One solution that is being used for this problem is the technique known as offloading. In recent years, several offloading support platforms have been proposed. This paper focuses on one of these platforms, called CAOS. Despite successfully performing offloading tasks, CAOS still has issues such as low scalability. In this study, we describe the process of migrating from CAOS to a new architecture based on microservices, highlighting the decisions and practices that were adopted on this journey.*

Resumo. *Os dispositivos móveis estão cada vez mais presentes no dia a dia das pessoas. No entanto, apesar da evolução das novas gerações de smartphones, a quantidade de informações e a complexidade dos procedimentos delegados a esses dispositivos, ainda impõem restrições ao processamento, principalmente relacionado ao consumo de energético. Uma solução que vem sendo utilizada para esse problema é a técnica conhecida como offloading. Nos últimos anos, várias plataformas de suporte ao offloading foram propostas. Este trabalho tem foco em uma dessas plataformas, denominado CAOS. Apesar de realizar tarefas de offloading com êxito, o CAOS ainda apresenta problemas como baixa escalabilidade. Neste estudo, descrevemos o processo de migração do CAOS para uma nova arquitetura baseada em microsserviços, evidenciando as decisões e práticas que foram adotadas nessa jornada.*

1. Introdução

Os dispositivos móveis, especialmente *smartphones*, são objetos cada vez mais presentes no cotidiano das pessoas. A evolução desses dispositivos permitiu a criação de diversas aplicações para auxílio em tarefas relacionadas à mobilidade urbana, troca de mensagens,

entre outras. Apesar da substancial melhoria das novas gerações de dispositivos móveis, a quantidade de informações e a complexidade de procedimentos delegados a estes dispositivos ainda impõe restrições para processamento de certas tarefas, principalmente em relação ao consumo de energia. Isto é especialmente problemático para aplicações móveis sensíveis a contexto, uma classe particular de aplicações móveis que utiliza informações obtidas do ambiente de execução do usuário, para adaptar seu comportamento em prol de benefícios para a experiência do usuário. Uma abordagem promissora para mitigar tal problema é a *Mobile Cloud Computing*(MCC). s Segundo [Dinh et al. 2013], MCC tem por objetivo provisionar um conjunto de serviços equivalentes aos da nuvem, adaptados à capacidade de dispositivos com recursos restritos, de modo a trazer melhorias de desempenho das aplicações ou economia de energia nos dispositivos. Ao longo dos últimos anos, soluções baseadas no conceito de MCC foram propostas para auxiliar na descentralização do processamento de dados e operações, diminuindo o consumo energético dos dispositivos [Artail et al. 2015, Ferrari et al. 2016, Liao et al. 2015]. Boa parte destas soluções baseia-se no uso de um técnica conhecida como *offloading* [Kumar and Lu 2010], onde processamento e dados são migrados de nós com baixo poder de processamento ou armazenamento, para dispositivos com mais recursos computacionais. Neste contexto, este trabalho foca em uma destas propostas: o CAOS[Gomes et al. 2017].

O CAOS permite a implementação, na plataforma Android, de aplicações móveis sensíveis a contexto com suporte ao *offloading* de processamento e dados contextuais. Experimentos com esta plataforma mostraram ganhos tanto no desempenho quanto na economia de energia em diversos tipos de cenários. Por restrições de espaço, os detalhes sobre a plataforma CAOS não puderam ser apresentados. No entanto, no estudo [Gomes et al. 2017] a plataforma CAOS é apresentada com maiores detalhes. O projeto arquitetural da versão inicial do CAOS criou um conjunto de serviços para dar suporte às decisões sobre o *offloading* de dados e processamento. Porém, estes serviços foram implementados com forte interdependência entre eles, criando um sistema monolítico com problemas para manutenibilidade, configuração e implantação. Tais características também levaram a um problema no suporte à escalabilidade do CAOS, permitindo somente a escalabilidade vertical, ou seja, aumentando recursos do nó onde a plataforma servidora seja implantada.

Para tratar questões previamente mencionadas em softwares monolíticos, uma abordagem recente que tem recebido muita atenção é o uso de microsserviços, em que aplicações são separadas em serviços independentes que interagem entre si para realizar as funcionalidades do sistema [Taibi et al. 2017]. A migração de softwares monolíticos para microsserviços apresenta diversos benefícios como melhores formas de gerenciamento de disponibilidade, escalabilidade de partes específicas do software, capacidade de utilização de diferentes tecnologias, redução do *time-to-market* e melhor compreensão da base do código [Fan and Ma 2017]. Este artigo apresenta um relato do processo de migração da plataforma CAOS para uma arquitetura de microsserviços, destacando o processo e decisões arquitetônicas que foram tomadas neste contexto de migração.

2. Processo de Migração da Plataforma CAOS

Na Figura 1 são apresentados os quatro passos do processo de migração seguidos para construção da nova versão do CAOS baseado em microsserviços: **(i)** Concepção e refinamento da proposta; **(ii)** Definição dos microsserviços; **(iii)** Definição das tecnologias;

e **(iv)** Desenvolvimento, testes, melhorias e concepção da nova arquitetura. Estas fases foram divididas em onze etapas, detalhadas a seguir.

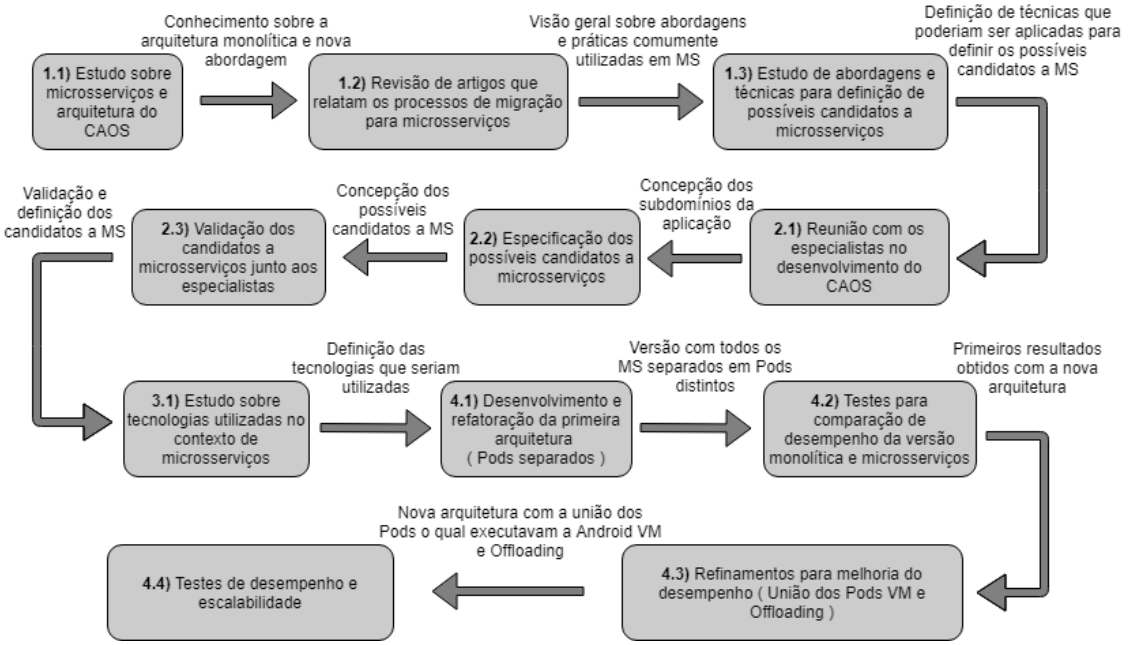

Figura 1. Processo utilizado na migração do CAOS

A etapa inicial do processo foi destinada à análise da versão inicial do CAOS. Nesta etapa, foram analisadas as funcionalidades, documentação, tecnologias, módulos e dependências que compõem a solução monolítica. Em conjunto, foram estudados conceitos e fundamentos da arquitetura microsserviços, buscando assimilar os benefícios que essa abordagem poderia proporcionar à solução CAOS. Com isso, foi possível obter uma visão inicial acerca dos benefícios que poderiam ser adquiridos com essa migração. Dentre eles, destacava-se uma escalabilidade mais eficiente, pois em momentos de alta demanda, seria possível escalar somente os recursos que estavam sendo sobrecarregados.

Na segunda etapa, foram revisados diversos artigos que relatavam a experiência de migração para a arquitetura de microsserviços, possibilitando uma visão geral das técnicas e práticas que poderiam ser utilizadas no contexto de migração proposto. Durante a revisão foi observado que os benefícios da arquitetura de microsserviços não são garantidos automaticamente, e só podem ser alcançados através de uma cuidadosa decomposição do software existente [Richardson 2018]. Nessa ocasião, foi percebido que para iniciar a migração para microsserviços, devem ser identificados os possíveis candidatos a microsserviços e que comumente é utilizada uma abordagem empírica para definição dos serviços da aplicação, ou seja, os serviços são definidos a partir do conhecimento dos *stakeholders* (desenvolvedores, projetistas e arquitetos) do software. No entanto, para realizar tais tarefas podem ser utilizadas técnicas, modelos e padrões de reengenharia já existentes na engenharia de software [Balalaie et al. 2015], [Taibi et al. 2017], [Fan and Ma 2017], [Dragoni et al. 2017], [Balalaie et al. 2016].

A terceira etapa teve por objetivo a escolha de uma abordagem a ser usada na refatoração do CAOS. Foram identificadas três possíveis abordagens: *Empirical Definition* (ED), ***S**ervice-oriented **P**rocess for **Re**engineering **a**nd **D**evops (SPReaD)* e *Do-*

main Driven Design (DDD). A ED permite a definição dos microsserviços baseado no entendimento acerca da arquitetura do sistema, possibilitando identificar os serviços que poderiam funcionar de forma individual [Taibi et al. 2017]. O SPReaD possibilita a reengenharia de sistemas legados, integrando os aspectos de DevOps para o direcionamento e concepção de serviços [Justino 2018]. O DDD entende o sistema como um grande domínio que pode ser dividido em subdomínios. Cada subdomínio corresponde a uma parte diferente do negócio. Os serviços devem ser desenvolvidos de acordo com os subdomínios da aplicação[Richardson 2018].

Baseado nos artefatos gerados nas etapas anteriores, na quarta etapa foi realizada uma reunião com três dos especialistas que participaram do desenvolvimento da plataforma CAOS. Nessa ocasião, foi realizada uma entrevista semiestruturada para melhor compreensão sobre a forma como o CAOS foi arquitetado, projetado e construído. Procurou-se identificar as dependências que haviam entre os componentes e quais as funcionalidades que a plataforma ofertava, possibilitando com isso a elaboração de uma breve concepção dos subdomínios da aplicação.

Na quinta etapa, foram discutidas as metodologias encontradas na etapa 1.3. Dentre as estudadas, a que mais se adequou ao contexto de migração proposto foi a *Domain Driven Design*. A escolha dessa técnica é justificada em razão da pré-divisão dos componentes da solução. Ainda em sua versão monolítica já era possível perceber claramente os subdomínios que representavam a ferramenta como um todo. Nesse sentido, o uso do DDD se mostrou bastante eficaz, pois diversos aspectos que essa metodologia possibilita eram almejados na nova solução. Esta metodologia tem como premissa a divisão do sistema em pequenos módulos ou subdomínios de um domínio principal, obedecendo os seguintes princípios: Favorecimento do reuso; Alinhamento do código com o negócio; Mínimo de acoplamento e Independência da Tecnologia.

Na sexta etapa, foi realizada uma nova reunião com os mesmos especialistas citados na etapa 2.1. Esse encontro visou apresentar e validar os candidatos a microsserviços obtidos com o uso do DDD no contexto da plataforma CAOS. Os candidatos a microsserviços foram validados de acordo com as funcionalidades que eles exerciam e de acordo com a função de cada um no contexto do domínio principal.

Além disso, como já relatado, a plataforma CAOS também permite o *offloading* de dados contextuais. Entretanto, como o esforço técnico no processo de migração era alto, a presente migração limitou-se a tratar somente dos microsserviços que possuíam relação com o *offloading* de processamento, não abrangendo as funcionalidades de aquisição e gerenciamento contextual. Entre os componentes que compõe a parte de *offloading* de processamento, foram identificados seis possíveis candidatos:

- **(I) Microsserviço de Autenticação e Descoberta**, responsável por identificar os dispositivos que estão se conectando a plataforma, e fornecendo um meio de acesso aos demais serviços da solução;
- **(II) Microsserviço de Monitoramento**, responsável por monitorar e registrar diversas métricas dos dispositivos móveis, além de receber e registrar as informações dos processos de *offloading* local e remoto. Essas métricas são utilizadas posteriormente para construção da árvore de decisão que é utilizada para se decidir sobre a viabilidade ou não do *offloading* de um método;
- **(III) Microsserviço de Tomada de Decisão**, que é o serviço que constrói as

árvores de decisão para cada método de aplicações suportadas pelo CAOS, levando em consideração as métricas dos *smartphones*, bem como os tempos de execução local e remota.

- **(IV) Microsserviço de *Offloading*** é responsável por receber os dados do método de um dispositivo e enviá-lo ao serviço de máquina virtual Android. Ao término do procedimento, o mesmo devolve o resultado ao dispositivo o qual solicitou o processamento;
- **(V) Microsserviço de Android Virtual Machine** é responsável por instanciar uma máquina virtual Android para realizar os processos de *offloading* requisitados pelo serviço de *Offloading*.
- **(VI) Microsserviço de Banco de Dados** encapsula um repositório responsável por registrar, fornecer e manter as informações geradas pelos demais microsserviços;

Na sétima etapa, foram estudadas e selecionadas ferramentas que pudessem auxiliar no desenvolvimento e implantação da nova arquitetura de microsserviços.Após a revisão realizada na etapa 1.2, cogitou-se a possibilidade de utilização das ferramentas `NetflixOSS`[1] (*Netflix Open Source Software Center*). Entretanto, percebeu-se que essas ferramentas criam muitas dependências, com impacto diretamente no desempenho, requisito chave para uma solução de *offloading*. Diante disso, analisaram-se diversas ferramentas que comumente eram utilizadas no desenvolvimento de microsserviços, a fim de identificar as que mais se adequavam ao cenário da plataforma CAOS e que pudessem fornecer as funcionalidades que são necessárias no ambiente de microsserviços. Entre as soluções encontradas, foram escolhidas o *Docker*[2] para conteinerização dos microsserviços e *Kubernetes*[3] para gerenciamento dos contêineres. Com isto, o projeto de refatoração do CAOS fez com que os microsserviços do lado servidor fossem transformados em imagens/contêineres do Docker, sendo gerenciados pelo Kubernetes.

Na oitava etapa, foi concebida a primeira versão do CAOS em microsserviços, batizada de CAOS MicroServices (CAOS MS). Nessa versão, a solução contava com todos os microsserviços separados e isolados em diferentes Pods no Kubernetes. Os módulos do lado cliente (dispositivo móvel) permaneceram inalterados, exceto por alguns ajustes na comunicação por conta de particularidades do Docker. Tal fato garante uma compatibilidade entre aplicações desenvolvidas para ambas versões da plataforma. Ressaltamos que somente os componentes do lado servidor passaram pelo processo de migração. Conforme pode ser observado na Figura 2, toda a comunicação entre o dispositivo móvel e os microsserviços ocorre de forma individual. Cada Pod representa um processo separado e individualizado dos microsserviços, que podem ser escalados e encerrados isoladamente. Um Pod é a menor unidade dentro de um cluster Kubernetes. Esse mecanismo permite a execução dos contêineres construídos no Docker[con 2018]. Vale ressaltar que em caso de falhas em algum microsserviço, os demais não serão afetados, e uma nova instancia do serviço onde ocorreu a falha será iniciada automaticamente pelo Kubernetes.

Na nona etapa, foram realizados testes de desempenho para validação da nova versão concebida. Foram realizadas comparações de desempenho entre a versão monolítica e a utilizando microsserviços. Para a realização dos testes de desempenho, foi

[1]URL: `https://netflix.github.io`, último acesso em 10 de Julho de 2019.
[2]URL: `https://www.docker.com`, último acesso em 10 de Julho de 2019.
[3]URL: `https://kubernetes.io`, último acesso em 10 de Julho de 2019.

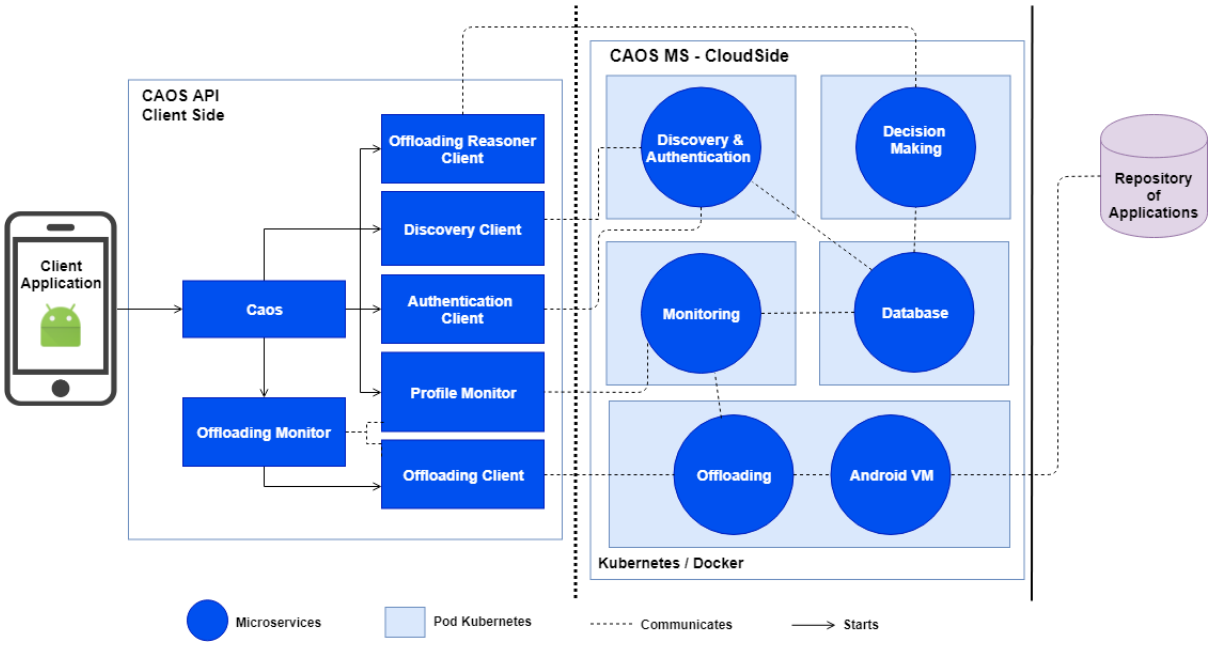

Figura 2. Arquitetura da plataforma CAOS MS.

instalada a aplicação `BenchImage` em dois *smartphones*. Essa aplicação já havia sido utilizada por [Gomes et al. 2017] em testes na validação da versão monolítica do CAOS. A aplicação BenchImage permite a aplicação de diversos filtros em imagens de diferentes resoluções, os quais demandam um razoável nível de processamento por parte dos *smartphones*. Dois tipos de filtros são fornecidos pela aplicação: `Cartoonizer` e `RedTone`. Além disso, as imagens disponibilizadas na aplicação variam de 0,3 megapixels até 8,0 megapixels, de modo que quanto maior a resolução, mais recursos computacionais são exigidos para realizar o processamento do filtro.

Nos experimentos com a aplicação BenchImage, foi utilizado o filtro *RedTone* em imagens com diferentes resoluções (0,3MP, 2MP e 8MP). Os experimentos foram repetidos 30 (trinta) vezes com todos os parâmetros citados e com ambos *smartphones*. Após a finalização dos testes, foi possível perceber que o tempo médio para realização do *offloading* na arquitetura de microsserviços ficou muito superior quando comparado a versão monolítica. O tempo médio para realizar o processamento, quando comparado com a versão monolítica, aumentou em 8,32 milissegundo(ms), equivalente a um aumento de 134%. Apesar dos diversos benefícios que foram proporcionados com a nova arquitetura, o tempo de execução invalidava a proposta, pois é um dos fatores primordiais para adoção de uma estratégia de *offloading*.

Na décima etapa, foram realizados novos procedimentos de testes para identificar o causador da lentidão encontrada na nona etapa. Nestes testes, foi capturado o tempo de execução em diferentes momentos do processo de *offloading*, com objetivo de se determinar os gargalos da execução remota dos métodos migrados. Então foi possível perceber que a separação dos microsserviços *Android VM* e *Offloading* em diferentes Pods no Kubernetes aumentava consideravelmente o tempo da execução do processo, pois após o microsserviço *Offloading* receber os argumentos da aplicação, era necessário acessar o serviço de descoberta do Kubernetes e transferir os dados recebidos para o Pod o qual

possuía o serviço de *Android VM*. Esse mesmo processo era repetido após o término da execução do processo do *offloading*, praticamente duplicando o tempo necessário para devolver o resultado ao dispositivo que havia requisitado o processamento. Diante desse fato, foi realizada a união dos microsserviços *Android VM* e *Offloading* em um único Pod. Essa alteração impossibilitou o escalonamento individual desses microsserviços. No entanto, ambos continuam com processos únicos e independentes (i.e., as alterações realizadas em qualquer um deles não impactam no funcionamento, manutenção e evolução do outro).

Na última etapa, após a união dos microsserviços *Android VM* e *Offloading*, os mesmos experimentos de desempenho realizados na etapa 4.2 foram repetidos. Com os resultados adquiridos foi possível perceber que o tempo para realização do *offloading* se manteve muito próximo em ambas arquiteturas e que em alguns casos, o tempo do CAOS MS foi levemente menor. Em seguida, foram realizados testes de escalabilidade com 20 (vinte) dispositivos reais, a fim de verificar a eficácia em um ambiente com um maior número de requisições. O teste de escalabilidade mostrou ganhos significativos em relação a antiga arquitetura, pois durante o teste uma maior quantidade de dispositivos conseguiu executar as tarefas de *offloading* com êxito e em menor tempo, comprovando a eficácia na escalabilidade da nova arquitetura.

3. Conclusões

Este artigo apresentou um relato da experiência sobre a migração da plataforma de *offloading* monolítica denominada CAOS, para uma arquitetura microsserviços. Com isso, pôde-se perceber que a migração para arquiteturas de microsserviços impõe diversos desafios ao time de desenvolvimento, pois exige alto conhecimento acerca do código fonte, comunicação entre componentes e arquitetura do software proposto, além de bom domínio em ferramentas e tecnologias que auxiliam na construção de softwares distribuídos. No entanto, com base no que foi apresentado, a nova arquitetura trouxe maior flexibilidade em diversos aspectos da solução, como: implantação, manutenção, escalabilidade e evolução, possibilitando que tais funções fossem realizadas em microsserviços isolados.

Destacamos que o processo apresentado no estudo visou evidenciar os passos que foram adotados para realizar a migração, não foi visado a criação de um modelo de processo para possíveis migrações de arquiteturas monolíticas para microsserviços. No entanto, as práticas adotadas podem servir de base para outras migrações que visam desempenho, como unir os microsserviços que necessitam de grande velocidade de comunicação em um único Pod.

Como trabalhos futuros, pretende-se conduzir novos experimentos relacionados aos aspectos arquiteturais da versão com microsserviços, evidenciando as melhorias que herdadas pelas funcionalidades presentes na solução. Além disso, serão refatorados os módulos de *offloading* de dados presentes na versão monolítica, para atuar na nova arquitetura em microsserviços.

Agradecimentos

Esta pesquisa foi parcialmente financiada pelos projetos INES 2.0, FACEPE PRONEX APQ 0388-1.03/14 e APQ-0399-1.03/17, CAPES 88887.136410/2017-00 e CNPq

465614/2014-0

Referências

(2018). Concrete — kubernetes. https://www.concrete.com.br/2018/02/22/tudo-o-que-voce-precisa-saber-sobre-kubernetes/. (Acessado em 13/07/2019).

Artail, A., Frenn, K., Safa, H., and Artail, H. (2015). A framework of mobile cloudlet centers based on the use of mobile devices as cloudlets. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, pages 777–784. IEEE.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pages 201–215. Springer.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., and Safina, L. (2017). Microservices: How to make your application scale. *arXiv preprint arXiv:1702.07149*.

Fan, C.-Y. and Ma, S.-P. (2017). Migrating monolithic mobile application to microservice architecture: An experiment report. In *AI & Mobile Services (AIMS), 2017 IEEE International Conference on*, pages 109–112. IEEE.

Ferrari, A., Giordano, S., and Puccinelli, D. (2016). Reducing your local footprint with anyrun computing. *Computer Communications*, 81:1–11.

Gomes, F. A., Rego, P. A., Rocha, L., de Souza, J. N., and Trinta, F. (2017). Caos: A context acquisition and offloading system. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 957–966. IEEE.

Justino, Y. d. L. (2018). Do monolito aos microsserviços: um relato de migração de sistemas legados da secretaria de estado da tributação do rio grande do norte. Master's thesis, Brasil.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56.

Liao, L., Qiu, M., and Leung, V. C. (2015). Software defined mobile cloudlet. *Mobile Networks and Applications*, 20(3):337–347.

Richardson, C. (2018). Pattern: Decompose by subdomain. https://microservices.io/patterns/decomposition/decompose-by-subdomain.html. (Acesso em: 11/04/2018).

Taibi, D., Lenarduzzi, V., Pahl, C., and Janes, A. (2017). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, page 23. ACM.