Análise de uma Aplicação de Modelagem Atmosférica em Nuvem e em Contêineres Utilizando Rastros

Lucas R. de Araujo[1], **Andrea S. Charão**[1], **João Vicente F. Lima**[1]
Haroldo F. de Campos Velho[2]

[1]Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria (UFSM)

[2]Instituto Nacional de Pesquisas Espaciais (INPE)

{lraraujo,andrea,jvlima}@inf.ufsm.br, haroldo.camposvelho@inpe.br

Abstract. *Exploring technologies such as the cloud and containers to execute high performance parallel applications might be beneficial, mainly to research in this area. Besides executing the applications, it is important to obtain metrics to observe the behavior of the application in each environment. In this context, traces were used to visualize the communication and analyze the performance of an atmospheric modeling application in these environments. The results point out a low overhead to the application executed and that the containers distribution might affect the performance.*

Resumo. *Explorar tecnologias como a nuvem e contêineres para execução de aplicações paralelas de alto desempenho pode ser benéfica, principalmente para a pesquisa nesta área. Além de executar as aplicações, é importante obter métricas para observar o comportamento da aplicação em cada ambiente. Nesse contexto, rastros foram utilizados para visualizar a comunicação e analisar o desempenho de uma aplicação de modelagem atmosférica nesses ambientes. Os resultados indicam baixo "overhead" para a aplicação executada e que a distribuição dos contêineres pode afetar o desempenho.*

1. Introdução

A computação em nuvem é um recurso com demanda crescente devido aos atrativos que oferece. Dentre eles, se destaca a elasticidade, que oferece o redimensionamento dos recursos de acordo com as necessidades do usuário. No campo de HPC (High Performance Computing), a nuvem da AWS (Amazon Web Services), através do serviço EC2[1] (Elastic Compute Cloud), traz recursos que beneficiam a execução dessas aplicações através de instâncias voltadas para computação, que oferecem maior capacidade de processamento, e serviços de rede com baixa latência e alta largura de banda na interconexão dos nós.

Outra tecnologia em ascensão são os contêineres, que se destacam pela leveza em sua virtualização, além da portabilidade. Essa característica engloba a questão de reproduzir um ambiente, da maneira facilitada, em diversas arquiteturas, o que pode acelerar a experimentação e a reprodutibilidade de pesquisas. Atualmente, o Docker [Merkel 2014]

[1]https://aws.amazon.com/pt/ec2/

é a ferramenta mais conhecida para utilização dessa tecnologia. Para computação distribuída, comumente utilizada em aplicações HPC, existe o Docker Swarm[2] para gerenciar a execução de uma aplicação em contêineres utilizando diversos nós computacionais.

Ao explorar esses diversos ambientes, se torna desejável executar a aplicação junto a ferramentas para coleta de dados, a fim de monitorar seu comportamento acerca de diferentes métricas. Uma das maneiras de analisar aplicações paralelas é através de rastros, que costumam mostrar os padrões de comunicação entre os processos, ao longo do tempo de execução. A utilização de ferramentas para obtenção de rastros é considerada vital para compreender o comportamento de aplicações em HPC [Eriksson et al. 2016].

No âmbito de HPC, existem aplicações para solução de problemas nas mais diversas áreas. Dentre essas áreas, existem as aplicações voltadas para previsão climática, como o BRAMS[3] (Brazilian developments on the Regional Atmospheric Modeling System) [Freitas et al. 2017]. O BRAMS é o modelo de previsão ambiental executado diariamente pelo CPTEC (Centro de Previsão do Tempo e Estudos Climáticos) do INPE (Instituto Nacional de Pesquisas Espaciais). De fato, o CPTEC-INPE foi o centro pioneiro em estruturar previsão ambiental operacionalmente.

Dessa forma, busca-se explorar a execução do BRAMS na nuvem AWS e, dentro da nuvem, os contêineres Docker, executando a aplicação junto a uma ferramenta para coleta de rastros. A proposta envolve avaliar a execução da aplicação em contêineres, em comparação ao ambiente de nuvem, além de avaliar diferentes maneiras de distribuir os contêineres através dos nós. Com os resultados, busca-se avaliar a viabilidade de utilizar contêineres para execução dessa aplicação, o que pode trazer benefícios para os pesquisadores. Além disso, os rastros serão analisados e seus dados descritos, para cada um dos casos de teste propostos, de maneira a complementar os resultados de desempenho.

2. Trabalhos Relacionados

Apesar da nuvem ser um ambiente virtualizado, [Evangelinos and Hill 2008] já a descreviam como um ambiente promissor para aplicações em HPC, através de execuções na nuvem AWS. Além disso, em um trabalho comparativo entre nuvens públicas, os autores indicam que a virtualização utilizada nessas nuvens, dentre elas a nuvem AWS, não adiciona nenhum *overhead* significativo aos tempos de execução, em comparação a resultados executados em um supercomputador [He et al. 2010].

Em relação aos contêineres, [Saha et al. 2018] destacam que o problema de segurança do Docker, relacionado ao acesso privilegiado, é minimizado em um ambiente de nuvem, visto que os usuários costumam ter esse tipo de acesso nesse ambiente. Os mesmos apresentam outra solução em contêineres, chamada Singularity [Kurtzer et al. 2017], que traz vantagens para execução de contêineres em centros de HPC. Eles utilizam essas duas tecnologias para execução de alguns *benchmarks*, para avaliar o desempenho e a granularidade da quantidade de contêineres e de nós.

Em outro trabalho que também explora Docker e Singularity, [Younge et al. 2017] utilizam Singularity para execução de *benchmarks* paralelos em um supercomputador Cray, em comparação com ambiente nativo e com o Docker na nuvem AWS. Os resulta-

[2]https://docs.docker.com/engine/swarm/

[3]http://brams.cptec.inpe.br/

dos mostram que os contêineres Singularity apresentam desempenho próximo ao nativo, enquanto na nuvem AWS existem mais problemas na escalabilidade da rede do que com *overhead* dos contêineres Docker, de acordo com o aumento no número de processos.

Diversos autores já estudaram maneiras de melhorar o desempenho do BRAMS, no âmbito computacional. As soluções variam de utilizar técnicas diferentes de programação, com OpenMP e OpenACC para arquiteturas *many-core* em parte do código [Silva Junior et al. 2017], até soluções focadas em melhorar o balanceamento de carga, através da virtualização de processos, comparando diversos algoritmos de balanceamento [Rodrigues et al. 2010]. Nesse último, os autores conseguiram reduções significativas no tempo de execução e destacam que o balanceamento de carga é muito importante para que haja escalabilidade em aplicações de modelagem climática.

Considerando todos os esforços realizados para melhoria da aplicação, busca-se nesse trabalho verificar o desempenho da aplicação BRAMS em um ambiente na qual ainda não foi executada, mas que pode ser muito favorável para que a pesquisa futura seja portável e disseminada rapidamente. Além disso, os trabalhos nas áreas de nuvem e contêineres serão uma base para verificar o comportamento desse novo caso nesses ambientes, com o reforço da coleta e análise de rastros para aumentar o escopo de informações.

3. Metodologia

Para execução da aplicação foram configuradas duas instâncias do tipo `c5.2xlarge`, na nuvem AWS. Cada uma dessas instâncias é composta por 8 vCPUs do processador Intel Xeon Platinum 8124M (3,00GHz), 16GB de memória RAM e largura de banda de rede de até 10 Gbps. Em ambas as máquinas, houve a instalação dos pré-requisitos e a compilação da aplicação, além da instalação do Docker para execução em contêineres.

A execução do BRAMS em ambiente de contêineres se dá através da criação de um Dockerfile[4], que é composto pelos passos de instalação dos pré-requisitos para compilação do modelo e a compilação do modelo em si. Foi construída a imagem a partir desse Dockerfile e, a partir da imagem, executa-se o contêiner que contém a aplicação.

Na parte relacionada a coleta e análise de rastros, diversas ferramentas foram consideradas para realização dessas tarefas. Ferramentas como Score-P e Scalasca, além de seus derivados para visualização dos dados coletados, foram descartadas devido a necessidade de instrumentação em código. Outra ferramenta considerada foi o EZTrace, porém seu funcionamento em contêineres apresenta falhas durante a execução.

Visto que as ferramentas já conhecidas apresentavam obstáculos para utilização, foi selecionada a ferramenta ITAC[5] (Intel Trace Analyzer and Collector). Além da compatibilidade com os ambientes de execução (nuvem e contêineres), autores destacam que a ferramenta apresenta baixo *overhead* e é de fácil utilização [Eriksson et al. 2016]. Sua instalação foi dividida em duas partes: a parte para coleta foi instalada nos ambientes de execução, nuvem e contêineres, e a de análise em ambiente local, um *desktop*.

Os casos de teste abrangem duas variáveis: número de processos e ambiente de execução. Os números de processos usados foram 1, 4 e 16, para todos os ambientes. Os

[4]https://docs.docker.com/engine/reference/builder/

[5]https://software.intel.com/content/www/us/en/develop/tools/trace-analyzer.html

ambientes usados foram a nuvem, com hipervisor *bare metal*, e os contêineres Docker. Se há mais de um processo, eles são divididos igualmente entre os nós.

As execuções diretamente na nuvem AWS serão identificados como `bare metal`. Já no ambiente Docker, a granularidade da quantidade de contêineres será avaliada: um caso com um contêiner por nó (1 contêiner para 1 processo e 2 contêineres para 4 e 16 processos), de granularidade alta, identificado como `docker-n`; outro caso com um contêiner por processo, de granularidade baixa, identificado como `docker-p`.

Cada um dos casos de teste foi executado 15 vezes, sendo 3 dessas com a instrumentação da ferramenta para coleta de rastros. Os dados mostrados na Seção 4 são a média dos valores obtidos para cada caso. Os casos de teste do Docker com um processo são considerados como um único, pois tanto para o caso `docker-n`, como para o caso `docker-p`, apenas um contêiner é utilizado.

4. Resultados

O caso da aplicação disponibilizado para execução está na página do BRAMS[6], descrito como *Small meteorological case*. A variação do tempo de execução, para os ambientes e números de processos executados, pode ser observada na Tabela 1. É possível acompanhar a diminuição do tempo de execução para todos os ambientes, de acordo com o aumento no número de processos, com bastante similaridade entre esses ambientes. Junto aos tempos de execução, é possível visualizar o *speedup* e a eficiência dos casos de teste, que são razoavelmente bons para 4 processos, porém apresentam uma queda ao executar a aplicação com 16 processos. A queda na eficiência é um pouco drástica, levando em consideração a quantidade de processos que as arquiteturas atuais são capazes de executar, mas pode estar atrelada a problemas de escalabilidade na rede da nuvem EC2, relatados anteriormente [Younge et al. 2017].

Caso de teste	Número de processos		
	1	4	16
bare metal	1040,95 (1,0 / 100%)	297,52 (3,49 / 87,46%)	137,91 (7,54 / 47,17%)
docker-n	1038,58 (1,0 / 100%)	301,68 (3,44 / 86,06%)	142,06 (7,31 / 45,69%)
docker-p		312,80 (3,32 / 83%)	155,61 (6,67 / 41,71%)

Tabela 1. Tempo de execução (s), acompanhando do *speedup* e eficiência.

Ainda na Tabela 1, destaca-se, principalmente para o caso de 1 processo, a proximidade entre o tempo de execução dos casos `bare metal` e `docker-n`/`docker-p`. Para o caso com 4 processos, o *overhead* é de cerca de 1,5%, para o caso `docker-n`, e cerca de 5% para o caso `docker-p`. Para o caso com 16 processos, o *overhead* é de cerca de 3%, para o caso `docker-n`, e cerca de 13% para o caso `docker-p`. Esse comportamento indica um aumento, no *overhead*, razoável para o caso `docker-n`, porém um aumento acima do *speedup* para o caso `docker-p`.

Em relação a comunicação, onde os dados foram obtidos através da coleta dos rastros, é possível visualizar a porcentagem do tempo de execução ocupada por funções MPI, para casos com vários processos, na Figura 1. Há um aumento nessa porcentagem

[6]http://brams.cptec.inpe.br/get-started/

de 4 para 16 processos, mas que está relacionada com a redução que há no tempo de execução, visto que os resultados são relativos. Apesar de a porcentagem da ocupação em funções MPI aumentar relativamente, é importante visualizar os dados absolutos. Na Tabela 2, esses dados são mostrados e indicam resultados variados, de acordo com o ambiente de execução considerado.

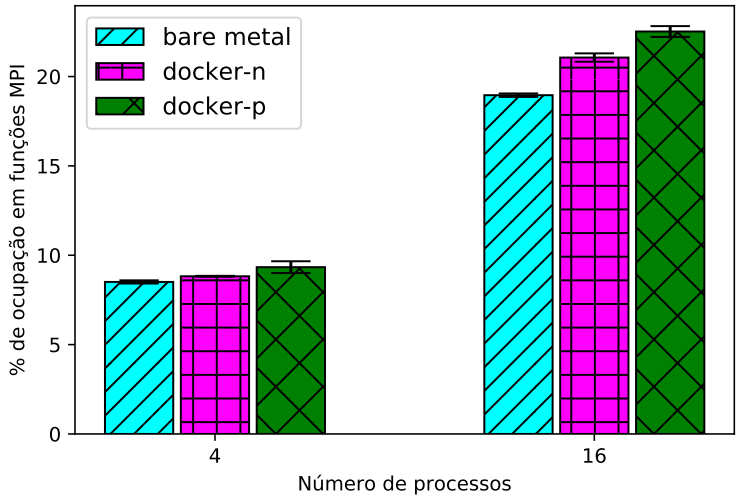

Figura 1. Porcentagem do tempo de execução ocupada por funções MPI.

Caso de teste	Número de processos	
	4	16
bare metal	25,28	26,13
docker-n	26,60	29,91
docker-p	29,18	35,04

Tabela 2. Tempo de execução (s) utilizado por funções MPI.

Ainda na Figura 1, tanto nas colunas da direita, quanto da esquerda, a porcentagem de tempo que as funções MPI ocupam aumenta do caso `bare metal` para o caso `docker-n` e desse para o caso `docker-p`. É um comportamento que acompanha o aumento no tempo de execução, visto na Tabela 1, mas com porcentagens maiores, o que indica que a comunicação é um fator que influencia bastante o *overhead* que foi visto nos tempos de execução, para os casos executados em contêineres. A porcentagem aumenta a medida que há o uso da rede de *overlay* (casos `docker-n` e `docker-p`) e um pouco mais com a diminuição da granularidade (caso `docker-p`).

De maneira geral, tanto os dados mostrados anteriormente (Tabela 1), quanto os dados da comunicação (Figura 1 e Tabela 2), indicam que o caso `docker-p`, com baixa granularidade, não é um caso que beneficia a aplicação. Seu *overhead* cresce acima do *speedup* no aumento de 4 para 16 processos, além do tempo de comunicação crescer quase o dobro em relação ao outro caso no Docker. Além de não ser ideal para os casos mostrados, em relação ao número de processos, o comportamento indica que essa configuração de contêineres não é escalável para casos com mais processos.

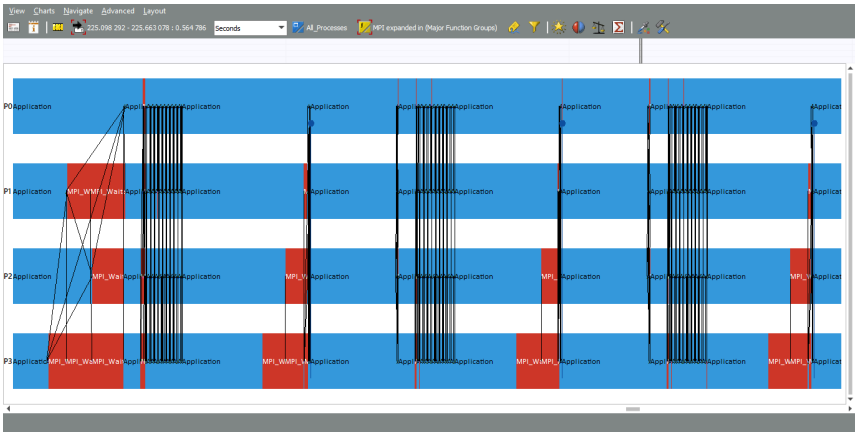

Figura 2. Rastro da aplicação (4 processos).

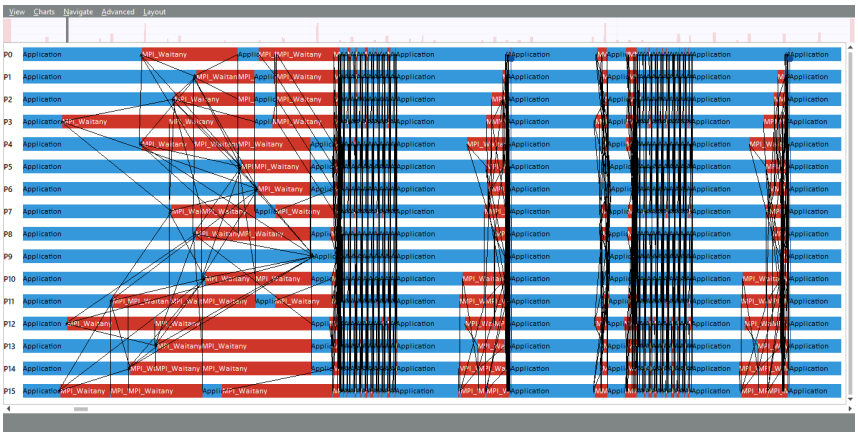

Figura 3. Rastro da aplicação (16 processos).

Um comportamento que se observou nos rastros, em relação as funções MPI utilizadas no código da aplicação e durante a execução, é que grande parte do tempo é ocupado pela função `MPI_Waitany`, para todos os casos de teste observados, com a devida proporção. Essa função caracteriza a utilização de comunicação não bloqueante, que permite que algum código não afetado diretamente pela comunicação seja executado até o ponto em que se necessita dos dados comunicados e então esse tipo de espera é invocada.

Parte desse comportamento pode ser observado nas Figuras 2, para 4 processos, e 3, para 16 processos, onde a parte em azul representa o processamento da aplicação, as linhas pretas representam a passagem de mensagem entre os processos, que acontece majoritariamente pelas funções `MPI_Isend` e `MPI_Irecv`, e a parte em vermelho representa a função `MPI_Waitany`. Essa fatia, desses 2 casos, representa o padrão da aplicação, que é o predomínio do tempo de ocupação do MPI pela função de espera.

O comportamento da aplicação em relação as esperas, resumido nessas figuras, pode estar atrelado ao desbalanceamento de carga, visto que os processos chegam em períodos de tempo diferentes em um mesmo ponto de código, ocasionando a espera. Essa ociosidade diminui o aproveitamento dos recursos computacionais disponíveis.

5. Considerações Finais

Foi realizado um estudo com a aplicação BRAMS sendo executada em um ambiente de nuvem e em contêineres, com o auxílio de uma ferramenta para coleta de rastros, a fim de monitorar os padrões de comunicação. O caso disponibilizado para execução é executado num período de tempo de simulação consideravelmente pequeno, principalmente se comparado com casos já mostrados na literatura [Rodrigues et al. 2019].

No presente trabalho, foi mostrado um baixo *overhead* para execuções em contêineres, principalmente com foco no caso de um processo, em que o tempo médio foi menor, e no caso `docker-n`, com múltiplos processos, os tempos de execução são semelhantes com os tempos obtidos no caso `bare metal`. Esse baixo *overhead* mostra que os contêineres podem ser um ambiente promissor para aplicações HPC, principalmente ao levar em consideração seus outros benefícios, como a leveza em sua virtualização e a reprodutibilidade que oferece.

Nesse contexto, é importante executar aplicações, ou casos de teste, de grande porte, de maneira a verificar se esse baixo *overhead* não se torna mais significativo para tempos de execução maiores. É importante lembrar que o Docker pode ser melhor utilizado como uma forma de portar a aplicação para testes, mais atrelado a pesquisa, enquanto as simulações com a aplicação final podem ser mais afetadas por esse *overhead*.

Em relação a granularidade, o caso `docker-n` apresentou melhores resultados que o caso `docker-p`. Foi observado que com o aumento do número de processos a diferença no tempo de execução aumenta entre os casos, assim como o *speedup* e a eficiência diminuem para o caso de baixa granularidade, o que mostra uma baixa escalabilidade para essa configuração de contêineres. Isso reitera o fato de que execuções com um contêiner por processo demonstravam baixo desempenho, nos *benchmarks* avaliados, em um trabalho anterior [Saha et al. 2018].

A comunicação, de maneira geral, acontece de forma rápida, sendo que o MPI passa a maior parte do tempo em espera, considerando a utilização de chamadas não bloqueantes. Essas chamadas sincronizam o código, quando necessário, visto que não há sincronização após a comunicação e sim quando necessita-se dos dados comunicados. A espera pode estar atrelada a um possível desbalanceamento de carga, visto que a aplicação já foi alvo de estudo em relação ao balanceamento de carga [Rodrigues et al. 2010]. Essa espera tem baixo *overhead* para o caso `docker-n`, mas para o caso `docker-p`, em relação ao caso `bare metal`, há um aumento significativo. Os resultados podem ser alterados em simulações de maior demanda computacional.

6. Agradecimentos

Este trabalho foi parcialmente financiado pelo projeto "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#162551-0000 488-9), no programa FAPERGS-CNPq PRONEX 12/2014. Autor HFCV agradece ao CNPq por suporte em projeto de pesquisa (Proc. 312924/2017-8).

Referências

Eriksson, J., Ojeda-May, P., Ponweiser, T., and Steinreiter, T. (2016). Profiling and tracing tools for performance analysis of large scale applications. *PRACE: Partnership for Advanced Computing in Europe*.

Evangelinos, C. and Hill, C. N. (2008). Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. In *In The 1st Workshop on Cloud Computing and its Applications (CCA)*.

Freitas, S. R., Panetta, J., Longo, K. M., Rodrigues, L. F., Moreira, D. S., Rosário, N. E., Silva Dias, P. L., Silva Dias, M. A. F., Souza, E. P., Freita, E. D., Longo, M., Frassoni, A., Fazenda, A. L., Silva, C. M. S., Pavani, C. A. B., Eiras, D., França, D. A., Massaru, D., Silva, F. B., Santos, F. C., Pereira, G., Camponogara, G., Ferrada, G. A., Campos Velho, H. F., Menezes, I., Freire, J. L. F., Alonso, M., Gacita, M. S., Zarzur, M. Z., Fonseca, R. M., Lima, R. S., Siqueira, R. A., Braz, R., Tomita, S., Oliveira, V., and Martins, L. D. (2017). The brazilian developments on the regional atmospheric modeling system (BRAMS 5.2): an integrated environmental model tuned for tropical areas. *Geoscientific Model Develpment*, 10:189–222.

He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010). Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, page 395–401, New York, NY, USA. Association for Computing Machinery.

Kurtzer, G. M., Sochat, V., and Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5):e0177459.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Rodrigues, E. R., Navaux, P. O., Panetta, J., Fazenda, A., Mendes, C. L., and Kale, L. V. (2010). A comparative analysis of load balancing algorithms applied to a weather forecast model. In *2010 22nd International Symposium on Computer Architecture and High Performance Computing*, pages 71–78. IEEE.

Rodrigues, L. F., Lima, S. T., Ruiz, R., Panetta, J., de Freitas, S. R., and Campos Velho, H. F. (2019). Parallel version for the BRAMS with Runge-Kutta dynamical core. Conference of Computational Interdisciplinary Science.

Saha, P., Beltre, A., Uminski, P., and Govindaraju, M. (2018). Evaluation of docker containers for scientific workloads in the cloud. In *Proceedings of the Practice and Experience on Advanced Research Computing*, pages 1–8.

Silva Junior, M. B., Panetta, J., and Stephany, S. (2017). Portability with efficiency of the advection of BRAMS between multi-core and many-core architectures. *Concurrency and Computation: Practice and Experience*, 29(22):e3959.

Younge, A. J., Pedretti, K., Grant, R. E., and Brightwell, R. (2017). A tale of two systems: Using containers to deploy HPC applications on supercomputers and clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 74–81.