Containers versus Máquinas Virtuais para Emulação de Controlo de Redes

Bruno Anjos, Nuno Morais, Paulo Lopes e José Legatheaux Martins

NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829–516 Caparica, Portugal
{b.anjos@campus., nm.morais@campus, paulo.lopes,
jose.legatheaux}@fct.unl.pt
http://di.fct.unl.pt

Resumo Dada a complexidade e as dificuldades associadas ao teste de protocolos distribuídos para controlo de equipamentos de rede num contexto real, é comum recorrer a emulação para teste dos mesmos. Os primeiros emuladores recorriam a máquinas reais, interligadas através de um sistema de emulação do tempo de trânsito e da perda de pacotes numa rede real [ModelNet].
Com o desenvolvimento das tecnologias de virtualização, seja esta através de máquinas virtuais ou de *containers*, tornou-se atractivo substituir as máquinas reais e os equipamentos de comutação por sistemas virtualizados. Um dos sistemas mais populares que seguem esta abordagem é o sistema MiniNet [MiniNet], para emulação de redes controladas com a abordagem Software Defined Networking e o protocolo OpenFlow. Outros sistemas, que utilizam outros protocolos e outros tipos de equipamentos de comutação, utilizam para o mesmo efeito máquinas virtuais [Cisco].
No quadro de um projecto em que se pretende testar protocolos de controlo de equipamentos de rede mais complexos que os switches OpenFlow, e utilizando protocolos distribuídos com semântica de falhas bem definidas, da classe dos usados em bases de dados distribuídas, coloca-se o problema de decidir qual a tecnologia de virtualização a utilizar: máquinas virtuais (VMs) ou *containers* implementados pelo sistema Docker.
Este artigo apresenta um estudo empírico comparativo das tecnologias: máquinas virtuais implementadas sobre o hipervisor ESXi da VMWare e containers Docker suportados quer no interior de VMs, quer em modo nativo. A execução de containers no interior de VMs, parecendo um contrasenso, não o é de facto pois a maioria das plataformas de grande escala (e.g., *clouds*) hoje acessíveis não disponibiliza acesso nativo aos utilizadores.

1 Introdução

1.1 XXXXXXXXXX

Explicar a razão de ser do trabalho, porque foi necessário, como se fez e dar uma breve panoramica dos resultados - usar parte do abstract que passará a ser mais resumido.

Dada a complexidade e as dificuldades associadas ao teste de protocolos distribuídos para controlo de equipamentos de rede num contexto real, é comum recorrer a emulação para teste dos mesmos. Os primeiros emuladores recorriam a máquinas reais, interligadas através de um sistema de emulação do tempo de trânsito e da perda de pacotes numa rede real [ModelNet].

Com o desenvolvimento das tecnologias de virtualização, seja esta através de máquinas virtuais ou de *containers*, tornou-se atractivo substituir as máquinas reais e os equipamentos de comutação por sistemas virtualizados. Um dos sistemas mais populares que seguem esta abordagem é o sistema MiniNet [**?**], para emulação de redes controladas com a abordagem Software Defined Networking e o protocolo OpenFlow. Outros sistemas, que utilizam outros protocolos e outros tipos de equipamentos de comutação, utilizam para o mesmo efeito máquinas virtuais [Cisco].

No quadro de um projecto em que se pretende testar protocolos de controlo de equipamentos de rede mais complexos que os switches OpenFlow, e utilizando protocolos distribuídos com semântica de falhas bem definidas, da classe dos usados em bases de dados distribuídas, coloca-se o problema de decidir qual a tecnologia de virtualização a utilizar: máquinas virtuais (VMs) ou *containers* implementados pelo sistema Docker.

Este artigo apresenta um estudo empírico comparativo das tecnologias: máquinas virtuais implementadas sobre o hipervisor ESXi da VMWare e containers Docker suportados quer no interior de VMs, quer em modo nativo. A execução de containers no interior de VMs, parecendo um contrasenso, não o é de facto pois a maioria das plataformas de grande escala (e.g., *clouds*) hoje acessíveis não disponibiliza acesso nativo aos utilizadores.

Foi então realizado um plano de trabalhos, visando definir os objectivos do projecto de investigação e estabelecer alguns *deadlines* e a respectiva alocação de tempo para cada tarefa.

[por discutir] → VALE A PENA COLOCAR AQUI O PLANO DE TRABALHOS?

Semana 1 (8/4/2018 : 15/4/2018)

1. Automatizar o deployment da aplicação de teste para executar múltiplos testes num quadro simples de um container por nó

Semanas 2 a 4 (15/4/2018 : 6/5/2018)

1. Estudar as tecnologias presentes e descobrir as diferenças fundamentais entre as mesmas
2. Formular uma hipótese sobre qual a raiz das diferenças de desempenho
3. Identificar os parâmetros e configurações a medir que permitam efetuar a comparação de resultados de forma rigorosa

Semana 5 a 7 (6/5/2018 : 27/5/2018)

1. Definir cenários (um ou mais servidores) e parâmetros de teste
2. Executar os testes
3. Analisar os resultados obtidos

Semana 8 a 9 (27/5/2018 : 10/6/2018)

1. Consolidar a análise dos dados obtidos
2. Elaboração do relatório

2 Máquinas virtuais versus *containers*

Bruno e Nuno vão adiantando Apresentação breve das duas tecnologias com pequeno esboço de comparação:

A virtualização, nomeadamente sob a forma de Máquinas Virtuais é uma tecnologia que se encontra presente em múltiplos sistemas em produção. Esta tem progredido maioritariamente devido á sua vasta utilização no contexto empresarial, as primeiras ocorrências desta surgiram na década de 1960 e atualmente ainda se encontram em desenvolvimento. Num entanto, recentemente surgiram novas tecnologias de virtualização com a introdução do *linux container* em 2013. Esta nova tecnologia trouxe consigo uma solução mais leve em termos de recursos exigidos, mas à custa de uma virtualização mais fraca e uma maior dependência do sistema do *host*. A vertente de emulação de containers é referida sempre num ambiente *linux* com o *Docker* como solução de *containers*. Esta decisão é tomada visto que as soluções para Windows e macOS passam pela implementação de um hipervisor sobre o sistema operativo correspondente que, neste caso de comparação entre as duas tecnologias, não faria sentido comparar uma implementação de containers que não corresponda a todo o potencial da solução.

2.1 Máquinas Virtuais

Num ambiente de virtualização por recurso a máquinas virtuais, o foco prinicpal cai sobre o hipervisor. Esta é a peça fundamental que se responsabiliza por ocultar o sistema *host* e disponibilizar apenas os recursos definidos à máquina virtual, virtualizando assim um subsistema do *hardware* que se apresentam como os recursos disponíveis para a mesma.

[por discutir] $\rightarrow$ **ACHO QUE ESTÁ BOM. CONFIRMA SFF**

2.2 *Containers*

A tecnologia dos *containers*baseia-se numa virtualização ao nível do sistema de operação através dos *control groups* e *kernel namespaces* presentes no *linux*, isto permite que várias instâncias de *containers*possam ser executadas sobre o mesmo kernel, evitando assim a camada de virtualização do *hardware* presente nas máquinas virtuais. Como referido anteriormente, a tecnologia Docker, utilizada como opção do ambiente de containers, domina atualmente o mercado no que toca a este tipo de virtualização.

2.3 Conclusões

Em suma as principais diferenças entre as duas opções de virtualização resumem-se no consumo e necessidade de recursos por instância, e no nível de abstração e independência do sistema *host*. Claro que outras diferenças como por exemplo o tempo de arranque, acabam-se por se traduzir numa maior necessidade de recursos, no caso da máquinas virtuais. Analisando assim estes dois aspetos, podemos afirmar que o *overhead* introduzido pelo hipervisor e pelo sistema operativo do *guest* é determinante na performance, o qual não existe nos *containers*, como já mencionadas em **??**.

2.4 XXXXXXXXXX

3 Benchmarks e metodologia de testes

Apresentar o que se pretendia avaliar, porque se fez um novo benchmark
Descrever o mesmo benchmark
Descrever sumariamente as variáveis e as medidas que se procuraram obter

Pretende-se avaliar a capacidade de emulação de um sistema distribuído de larga escala que efetua uma grande quantidade de trocas de mensagens periodicamente. Para tal foi criado um *benchmark* baseado na análise do tempo de execução do ciclo principal da aplicação. Fazendo variar o débito e o número de elementos do sistema distribuído é possível analisar a capacidade de ambas as tecnologias de emular de uma rede virtual e também observar quais os limites desta.

Foi desenvolvida uma aplicação em python cujo objetivo consiste na inserção periódica de tuplos numa base de dados mysql local e periodicamente enviar os mesmos agroupadamente para serem inseridos em bases de dados remotas, replicando assim a base de dados.

[por discutir] → ADICIONEI ESTA PARTE QUE DIZ O PORQUE DE NAO UTILIZARMOS UM BENCHMARK PRE-FEITO

Este formato de testes foi construído de raíz, pois o que foi verificado no trabalho relacionado (ver **??**), é que por norma os testes de *performance* entre máquinas virtuais e containers são a um nível local e não implicam qualquer tipo de comunicação. Visando este projecto a comparação de *performances* de diferentes ambientes a um nível distribuído, foi então desenhada uma aplicação que representasse um sistema desse mesmo tipo.

3.1 Funcionamento da aplicação

- Seja D o débito da aplicação, a razão entre a quantidade de tuplos inseridos e uma unidade temporal. D = $\frac{msg}{minuto}$
- Seja AG o fator de agregação das mensagens enviadas para as bases de dados remotas
- Seja N o número de instâncias que compõe o sistema distribuído

O fluxo da aplicação consiste em três estados principais:

1. Sincronização

   A fase de sincronização possui como objetivo a coordenação das múltiplas instâncias da aplicação. Esta consiste no estabelecimento de ligações entre os diversos componentes do sistema. Após estabelecidas as ligações, é inserida uma entrada numa tabela de sincronização pertencente á base de dados elemento cujo endereço IP é o menor (*master node*). Posteriormente são efetuadas listagens dessa tabela periodicamente, no momento em que o número de entradas for igual ao número de elementos do sistema distribuído é concluído que todos os elementos do sistema estão prontos para prosseguir para a seguinte fase de execução.

2. Ciclo principal

   No ciclo principal periodicamente (de acordo com D), será gerado um número aleatório de 0-100 e gerada uma sequência aleatória de carateres. Um tuplo composto pelos elementos referidos anteriormente será inserido na tabela da base de dados local. Após AG inserções locais serem executadas, são inserido os mesmos tuplos agregados nas restantes N - 1 bases de dados. Cada instância da aplicação irá executar estas inserções, resultando na tabela referida anteriormente replicada em N bases de dados.

3. Dessincronização

   Na fase de dessincronização todos os elementos do sistema distribuído irão remover o tuplo inserido durante a fase de sincronização (presente na tabela do *master node*), em seguida irá efetuar a listagem dessa tabela até que esta esteja vazia. Nesse momento é concluída a fase de dessincronização.

4 Resultados dos testes e sua discussão

Dada a complexidade e as dificuldades associadas ao teste de protocolos distribuídos para controlo de equipamentos de rede num contexto real, é comum recorrer a emulação para teste dos mesmos. Os primeiros emuladores recorriam a máquinas reais, interligadas através de um sistema de emulação do tempo de trânsito e da perda de pacotes numa rede real [ModelNet].

Tal como discutido em **??**

5 Trabalho relacionado

Apresentar outros estudos assim como técnicas usadas fazer referência breve a ModelNet, PlanetLab, Mininet, ... estudos parecidos

6 Conclusões e trabalho futuro

Dada a complexidade e as dificuldades associadas ao teste de protocolos distribuídos para controlo de equipamentos de rede num contexto real, é comum recorrer a emulação

para teste dos mesmos. Os primeiros emuladores recorriam a máquinas reais, interligadas através de um sistema de emulação do tempo de trânsito e da perda de pacotes numa rede real [ModelNet].

[por discutir] → DEPOIS VÊ SE CONCORDAS

Houveram alguns precalços nomeadamente na utilização do motor de base de dados, pois sabendo que os *benchmarks* são testes de performance, há uma necessidade de otimizar o motor de base de dados de forma a introduzir o menor *overhead* possível, pois o objectivo em questão é verificar a performance consoante as variáveis mencionadas em **??**. Foi discutido o aumento do número de *threads* de escrita na base de dados visando melhorias ao nível do paralelismo, mas a ideia acabou por ser descartada, devido ao número de núcleos lógicos atribuído a cada máquina virtual. O maior imprevisto que pode ser tido em conta num trabalho futuro, surgiu aquando dos testes nas máquinas virtuais, em que, na fase de sincronização (ver **??**), cada máquina virtual demorava bastante tempo (por volta de 5 minutos) a abrir as conexões para as restantes instâncias. Após algum aprofundamento da questão, chegou-se à conclusão de que estava relacionado com a resolução de nomes das restantes máquinas. Foram implementadas algumas possíveis soluções, tais como, desativar a resolução de nomes do motor da base de dados e adicionar os endereços e os nomes ao ficheiro de resolução de nomes utilizado pelo sistema operativo, mas em nenhuma das soluções se verificou uma melhoria na situação em questão. De qualquer das formas esta questão não foi mais aprofundada pois a fase que dita a performance de um determinado teste (ciclo principal, ver **??**) não é influenciada.

Referências

1. B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.