Novas tecnologias para aplicações distribuídas

Tecnologias em pleno uso nas aplicações de sistemas distribuídos.

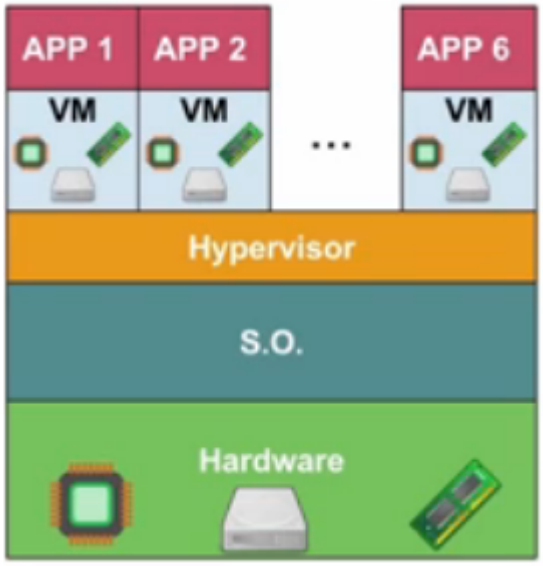
Assuntos abordados

- Virtualização
- Conteinerização
- Micro Serviços
- Serverless

Virtualização

Objetivo: Otimizar o uso do hardware, permitindo o seu compartilhamento entre várias instâncias de sistemas operacionais.

Até seu advento a cada nova aplicação era necessário uma nova máquina. Assim ela otimiza o uso do hardware para se ter numa mesma máquina diversas instâncias.

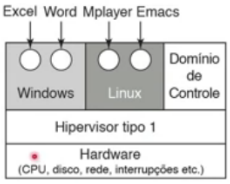


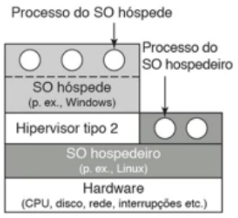
Benefícios

- Isolamento de diferentes SOs num mesmo hardware
  instâncias independentes uma das outras.
- Economia de espaço, energia, custo de hardware.
  requer menos máquinas.
- Ajuda aos desenvolvedores para testar aplicações em diferentes SOs
  pode-se criar diferentes máquinas para testes e depurações.
- Facilidade de migração das máquinas
  pode-se mudar de hardware sem parar o ambiente.
- Balanceamento de carga
  O ambiente de virtualização verifica a carga de uma máquina física para realizar a mudança de máquina.

Problemas

- Cada aplicação necessita de uma instância completa de SO
  Consumo de espaço e armazenamento.
- Custo adicional na manutenção de cada máquina virtual
- Consumo de recursos computacionais com cada instância de SO



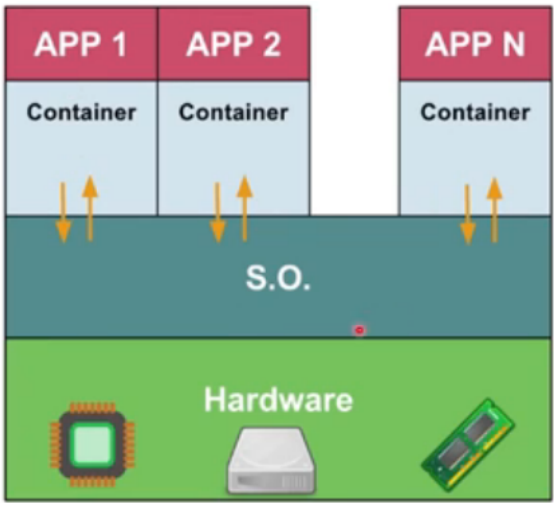


Modelo hipervisor tipo 1: limitado e sem interação direta. Usado em empresas de médio e grande porte para virtualização de servidores.

Modelo hipervisor tipo 2: no hardware se tem um sistema operacional tradicional (hospedeiro) que executa o ambiente virtual. Exemplo: VirtualBox.

Conteinerização

- Surge para resolver o problemas da Virtualização
  vem substituindo o modelo de virtualização.
- O container funciona junto ao SO base de forma a dividir as funcionalidades do mesmo entre as aplicações.
  A ideia é ter uma única instância de SO, o qual divide todas as funcionalidades com as aplicações que estarão sendo executadas no ambiente de virtualização.
- Não há mais um SO para cada aplicação
- Apenas um SÓ compartilhado entre todas as aplicações
- Reduz o custo de manutenção e infraestrutura
  Gerencia apenas uma única máquina e há apenas o gerenciamento das aplicações.
- Mais leve e mais rápido de subir
  A carga é muito mais rápida que a carga de um SO.
- Surgiu da necessidade de isolar duas ou mais aplicações.
  Basicamente se cria contêineres para realizar o isolamento de aplicações.



O Docker

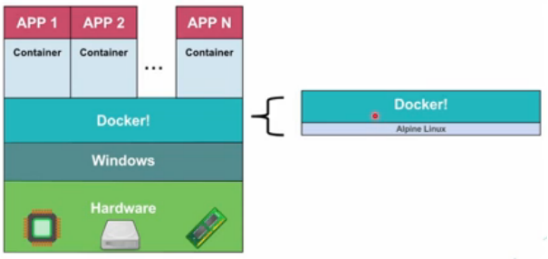
- Primeira implementação do conceito de conteinerização
- **Docker engine** - Responsável pela interação com o SO Base.
  realiza interação entre o SO da máquina e os contêineres

- **Docker compose** - Define e orquestra múltiplos containers.
  realiza a orquestração: verifica se um contêiner falhou e caso tenha falhado é levantado outro contêiner com as mesma caracteristicas da aplicação
- **Docker Swarm** - Possibilita a criação de um cluster com múltiplos Docker Engine.
  Nesse ambiente pode-se fazer o gerenciamento do cluster para caso um contêiner falhe, seja inicializado outro.
- **Docker Kubernets** - Ambiente de gerenciamento de cluster mais utilizado.
  Realiza o mesmo que o compose e o Swarm porém com muito mais recursos.
- **Docker Hub** - Repositório de imagens para os containers.
  As imagens servem para que configure o ambiente. Também se tem bases para construção das próprias imagens.
  imagens?
- **Docker Machine** - Permite o gerenciamento do Docker em um host virtual.
  Possibilita que se crie máquinas de conteinerização. Também permite a disponibilização dos contêineres na nuvem.

Instalação do Docker

- Docker for Windows - Usa Hyper-V da Microsoft
  - Atenção: Desabilita todos os outros ambientes de virtualização!
  - https://store.docker.com/editions/community/docker-ce-desktop-windows
- Docker Toolbox - Usa o Virtualbox(descontinuado!)
  - https://download.docker.com/win/stable/DockerToolbox.exe
- Verificar o sucesso da instalação
  - docker run hello-world

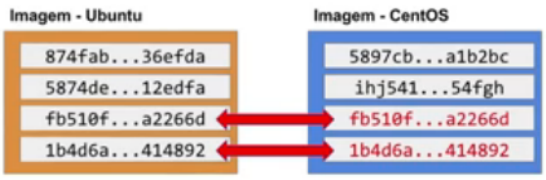
Docker for Windows



Layered File System

- Uma imagem é composta uma ou mais camadas
  onde cada camada tem uma pequena parte da imagem.

- Camadas podem ser reaproveitadas
  Bibliotecas em comum, por exemplo, podem ser compartilhadas através de camadas. Tudo que é comum aos contêineres fica em camadas.
    - Ex: Ubuntu e CentOS compartilhando camadas
- Economiza tempo e espaço
- Camadas são somente leitura
  Para garantir que qualquer outra imagem que precise de determinada camada não tenha problemas caso alguma outra imagem altere.
- Quando um container é criado, uma camada com permissão de escrita é criada.
  Camada que contem o que foi adicionado a imagem original.



Comandos Docker

- docker run -d -p 8080:80 --name site dockersamples/static-site
    - Executa um container desvinculado do terminal(-d), mapeando a porta 8080 da máquina na porta 80 do container e chamando o container de site
- docker search --no-trunc <padrão>
    - Busca por imagens no Docker Hub com o padrão especificado, sem truncar a descrição
- docker run -d -P -e AUTHOR="Fulano" dockersamples/static-site
    - Seta uma variável ambiente no container.
- docker run -it -v "C:\temp:/var/www" ubuntu
    - Cria um volume(/var/www) no containêr apontando para c:\temp na máquina

Criando uma imagem

- Criar um arquivo Dockerfile (extensão .dockerfile)
- Conteúdo

```
FROM picoded/ubuntu-openjdk-8-jdk
MAINTAINER Weslley Lima
COPY loja /var/www
WORKDIR /var/www/target/test-classes
ENTRYPOINT java -classpath /var/www/src/main/webapp/WEB-INF/classes:/var/www/src/main/webapp/WEB-INF/lib/*:. br.com.alura.loja.Servidor
EXPOSE 8080
```

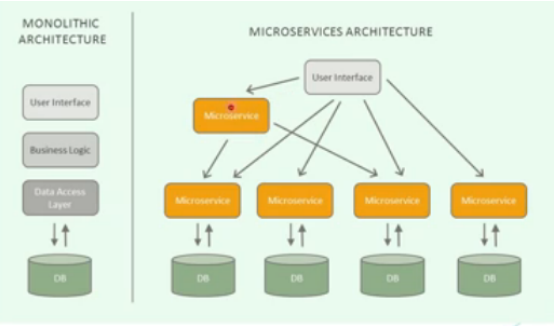
Criando uma imagem

- Construir a imagem no Docker hub
    - docker build -f Dockerfile -t weslley/loja .
- Criar um container a partir da imagem criada
    - docker run -it -p 8080:8080 weslley/loja
- Testar aplicação!

Arquitetura de Microservices

- Estilo de desenvolvimento que segmenta aplicativos em micro serviços fracamente acoplados e desenvolvido por equipes multifuncionais.
- Cada serviço possui:
    - Seu próprio processo de deploy.
    - Sua própria tecnologia.
    - Seu próprio armazenamento.
- Serviços comunicam-se através de chamadas remotas.
- Adequado para o desenvolvimento de sistemas complexos.

Monolítico X Micro Serviços



Arquitetura monolítica: Limitações

- Aplicações grandes e complexas
- Dificuldade de modificação e atualização
- Baixa resiliência: Um bug em uma parte pode comprometer toda a aplicação.
- Diferentes partes da aplicação possuem diferentes requisitos de recursos computacionais.
- Aplicação com vários “donos".
- Dificuldade de testar novas funcionalidades.
- Uso de uma única linguagem de programação

Arquitetura de microservices

- Modelo antigo: Usado no UNIX
- Executa algo de forma bem feita
- Executa independentemente

Micro Serviços: Razões para uso

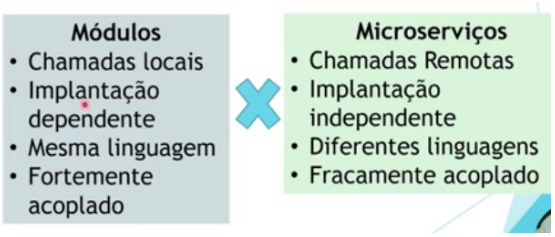
- Número elevado de desenvolvedores para uma aplicação.
- Grande número de linhas de código.
- Frequente liberação de releases.
- Incremento gradual de funcionalidades da aplicação.
- Troca frequente na equipe de desenvolvedores.

Micro Serviços: Alerta!

- Acrescenta complexidade.
- Não adequado para aplicação de pequeno e médio porte.
- Complexidade adicional compensada em grandes projetos com o passar do tempo.

Módulos X Micro Serviços

- Modularidade sempre foi um objetivo
- Micro Serviços só reforça esse princípio



SOA e Microsserviços

- Micro Serviços são uma forma específica de implementar a arquitetura SOA.
- Todos os princípios do SOA são aplicados.
- SOA: SOAP + HTTP
- Microserviços: REST + HTTP

Micro Serviços: Princípios

- Times independentes.
- Bancos de dados independentes.
- São sem estado(stateless).
- Uso de containers: empacotamento e rápida inicialização.
- Uso de nuvens.

Micro Serviços: Mudanças

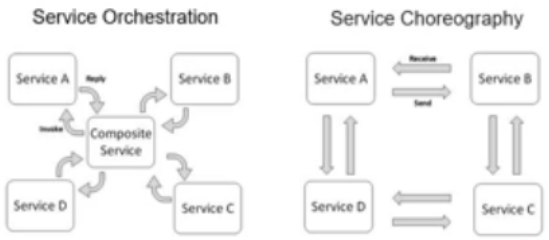
- Complexidade adicional em função de cada micro serviço possuir seu próprio banco de dados
- Compartilhamento de BD introduz
- acoplamento: Ruim!
- Troca de dados entre Micro Serviços sempre através de troca de mensagens.
- Provável existência de dados duplicados.

Orquestração X Coreografia

- Orquestração: Um serviço central coordena a chamada de outros serviços. Ponto de

falha. Usando em sistemas monolíticos.

- Coreografia: Cada serviço reage a eventos que ocorrem. Utiliza comunicação publicar/assinar. Usado em micro serviços.



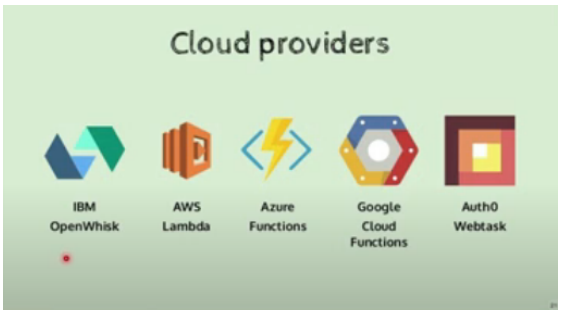
Micro Serviços: Dificuldades

- Mais pontos de falhas
- Mais containers/VMs/Hosts
- Mais bancos de dados.
- Log espalhado entre vários pontos
- Monitoramento

Serveless

- Também conhecida como FaaS(Function as a Service)
- Vantagens:
  - Não é necessário gerenciar ou provisionar servidores
  - Paga apenas pelo uso. Não há custo para os momentos de ociosidade.
  - Altamente escalável
  - Grande disponibilidade e tolerância a falhas
  - Suporte a diversas linguagens e frameworks

Serverless: Provedores



Comparativo Serviços AWS

	Characteristic	*AWS Serverless Cloud*	*AWS Serverful Cloud*
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.0000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible