**Sistemas de Computação em Cloud**

### First generation batch processing: Map-reduce

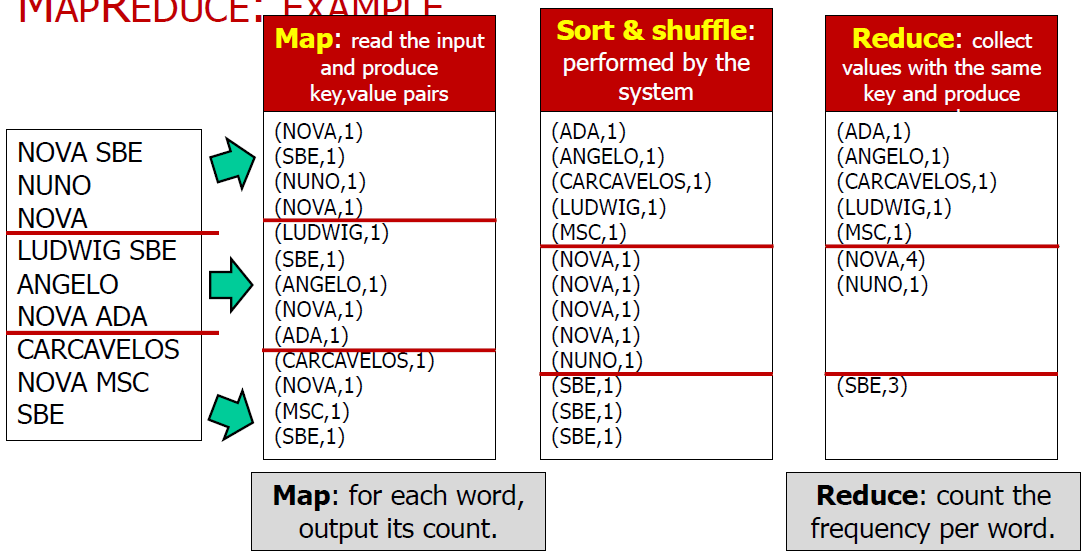
Permite expressar cálculos simples que estávamos tentando realizar, mas esconde os detalhes confusos de:

* paralelização
* tolerância a falhas,
* distribuição de dados
* balanceamento de carga em uma biblioteca

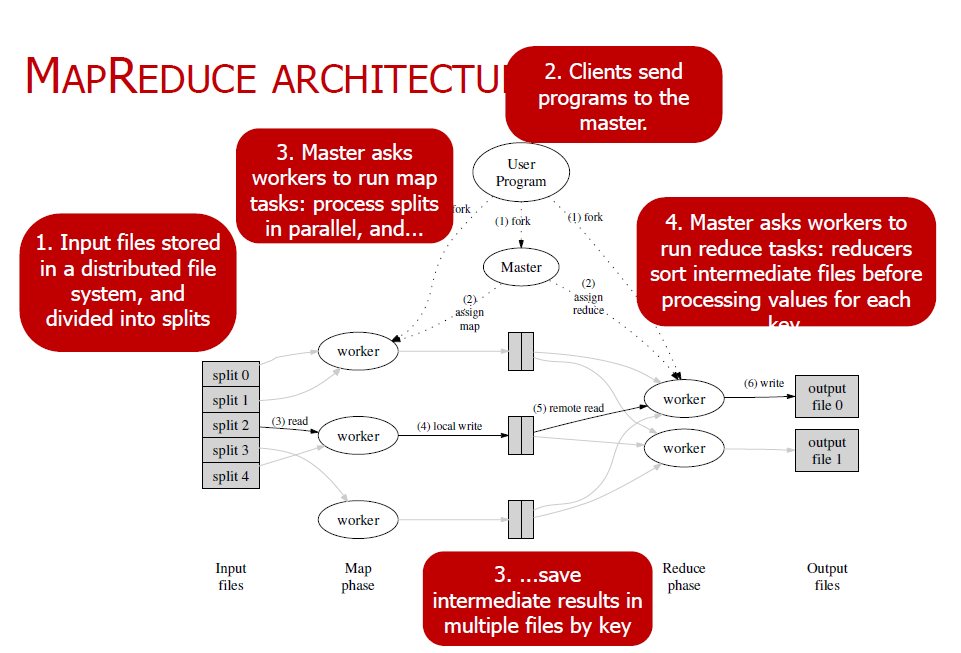
Um modelo de programação e uma implementação associada para processar grandes conjuntos de dados

## OverView

1. Lê sequêncialmente muita informação
2. Map phase – extrai a info. importante
3. Agrupa por chave (Sort and Shuffle do output da Map phase – por frequência da palavra)
4. Reduce phase - Aggregate, summarize, filter or transform
5. Dá o resultado



# Arquitetura



## Master Node

O **Master node** coordena a execução:

* Task status: (idle, in-progress, completed)
* Idle Tasks: as tarefas ociosas são agendadas à medida que os trabalhadores ficam disponíveis
* Quando uma tarefa de mapa é concluída, ela envia ao mestre a localização e os tamanhos de seus arquivos intermediários, um para cada redutor
* O master envia essas informações para os redutores
* O mestre pinga os trabalhadores periodicamente para detectar falhas

## Worker

O nó do trabalhador executa tarefas de mapeamento ou redução, conforme solicitado pelo coordenador.

## Handling Faults

**Map worker failure**

* Após a detecção da falha de um trabalhador, as tarefas do mapa são reiniciadas em diferentes trabalhadores

**Reduce worker failure**

* A tarefa de redução é reiniciada em outro trabalhador

**Stragglers (slow workers)**

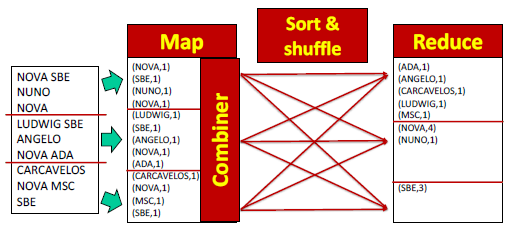
* Se uma tarefa estiver demorando muito para ser concluída, ela será iniciada em outro trabalhador. E usado o primeiro resultado.

**Falha mestre**

* A tarefa MapReduce é abortada e o cliente é notificado

# Map reduce Improvement – Combiner

Combiner permite pré-agregar valores no mapper. Normalmente o mesmo que a função de redução



# Desvantagens

O MapReduce requer cálculos complexos para serem divididos em sucessivos MapReduce jobs

Esses programas apresentam alta latência devido a:

* + precisa ler e gravar arquivos
  + replicação do sistema de arquivos fundamental (para gravações)
  + um trabalho deve terminar antes que o próximo possa ser iniciado

O Apache Spark aborda essas limitações.

### Second generation batch processing: Spark

Apache Spark fornece processamento distribuído na memória e tolerante a falhas.

* Programas Spark:
  + múltiplas transformações de dados encadeadas
  + usam um modelo de programação funcional de alto nível;
* O Spark define uma estrutura de dados de coleta distribuída: **Resilient Distributed Dataset (RDD)**

# Data Models e API’s

## RDDs são dados imutáveis

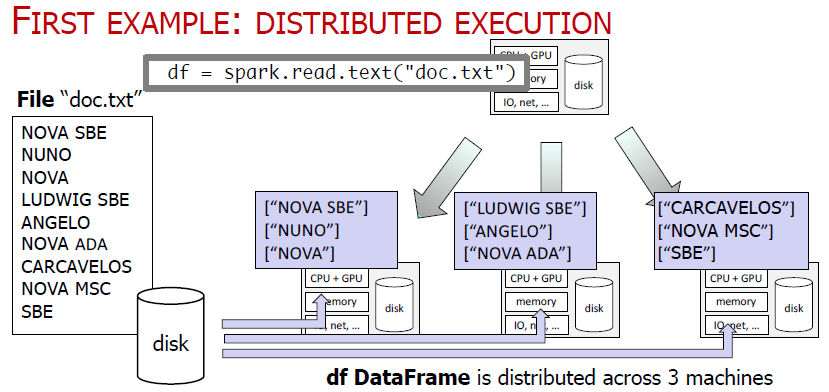
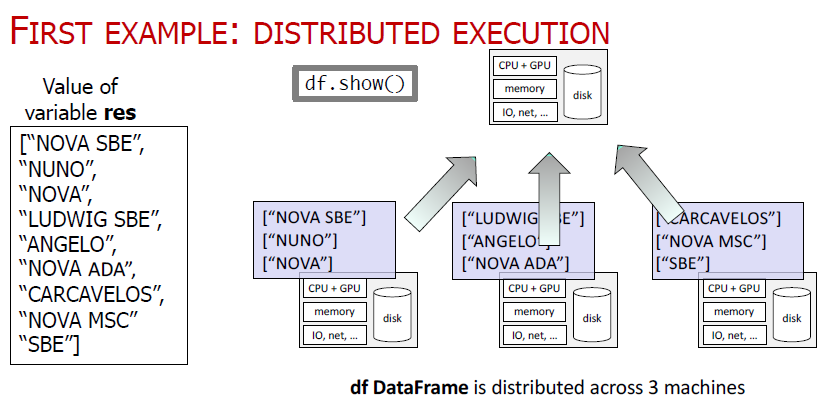
* + um RDD é uma coleção imutável de tuplos de dados;
  + distribuído (particionado) em muitos nós;
  + após uma falha os RDDs podem ser recriados automáticamente e eficientemente a partir das dependências.

## Quadros de dados do Spark

* + DataFrames - coleções distribuídas de dados que são agrupadas em colunas com um nome.
  + DataFrames podem ser vistos como RDDs com um esquema que nomeia os campos das tuplos fundamentals.

## Spark SQL

* + SQL para especificar computações
    - Programas que usam SQL/DataFrames são convertidos em programas Spark.
    - Programas são otimizados para serem executados com eficiência.
    - Usadas técnicas utilizadas em sistemas de BD.
    - Bibliotecas para algoritmos de análise avançada, como processamento de gráficos e aprendizado de máquina.

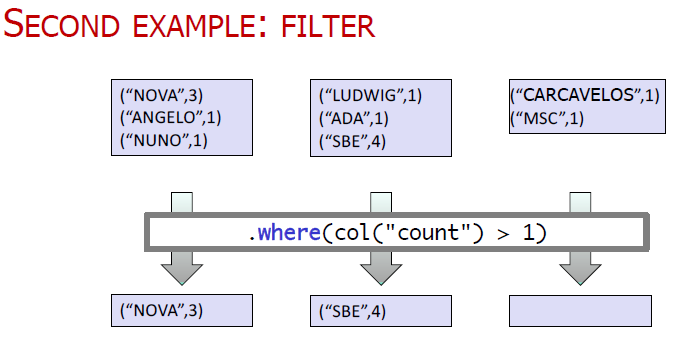
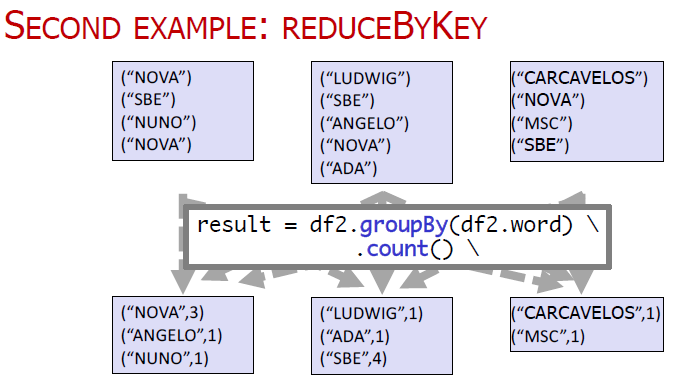
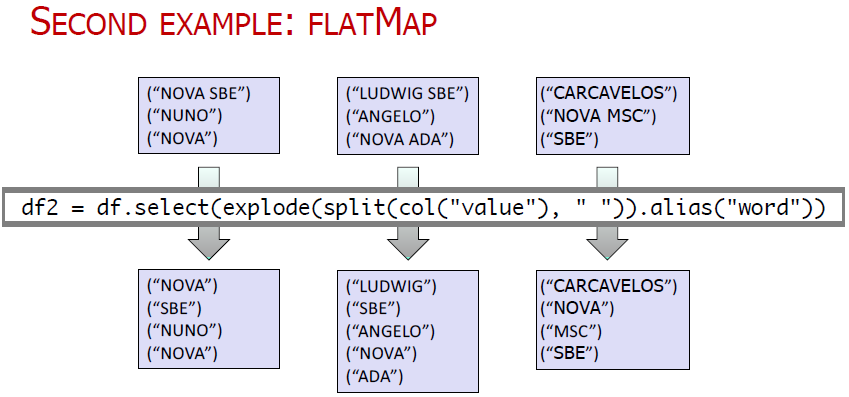
 

# Programming model

Os programas Spark Dataframe descrevem o fluxo de transformações que cria um DataFrame a partir de outro, geralmente em várias etapas. Programas Spark, codificam as dependências entre os vários DataFrames (e RDDs fundamentals) - gráfico de linhagem.

**Wide-Dependencies** - produzidas quando uma partição RDD depende de várias partições armazenadas em nós diferentes (alto custo de bandwidth no groupByKey)

**Narrow-dependencies** - produzidas quando uma partição RDD depende de dados que estão no mesmo nó



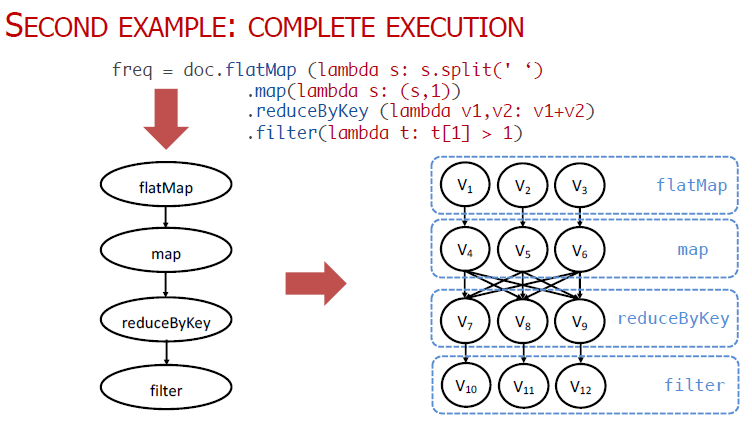
# Programming and execution model

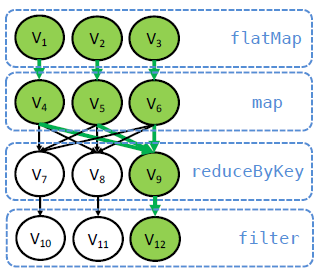
Programas DataFrame 🡪 convertidos 🡪 programas RDD, que envolvem:

* Transformações: RDD -> RDD
* Ações: RDD -> Resultado (disponível diretamente para o aplicativo cliente)

## Execução

A execução consiste em aplicar as transformações em todas as partições de um RDD em paralelo. O desempenho é melhor quando o resultado de uma partição RDD não requer dados de partições RDD de entrada localizadas em nós diferentes (ou seja, evita embaralhamentos)





# Fault tolerance

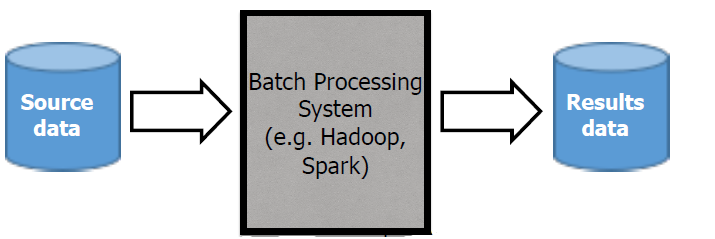
Sparks lida com falhas de nó recalculando partições perdidas, usando informações de linhagem.

* Otimizado por RDDs intermédias persistentes.

No exemplo, se V9 for persistido, muitos recálculos serão salvos.

### Stream processing (Big data / batch processing)

**Objetivo:** Executar computação sobre dados e produzir resultado

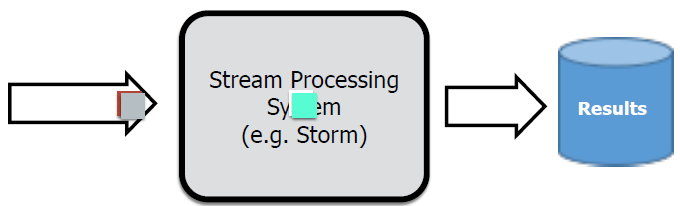


Exemplo: Produzir informação sobre o trânsito com base na informação recolhida dos telemóveis dos users

# Processing models

## Continuous

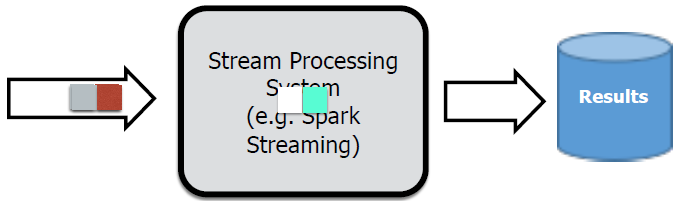
* Cada tuplo é processada à medida que chega
* O sistema de processamento pode manter o estado para executar **window computation** e **incremental computation**



## Mini- batches

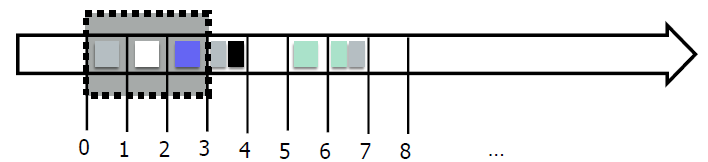
Tuplas recebidas para cada **X** ms agrupadas em um mini-lote

* Processar mini-batches
* O sistema de processamento pode manter o estado para executar **window computation** e **incremental computation**



# Windowing

Ao fazer o stream processign, geralmente é interessante calcular os resultados com base nos dados de um determinado intervalo, mas calcular os resultados com mais frequência do que o intervalo de tempo — por exemplo, processar dados dos últimos 3 minutos, mas produzir resultados a cada minuto. O sistema para processamento de fluxo suporta a definição de janelas de tempo deslizantes.



### Virtualization

A virtualização consiste na criação de recursos computacionais simulados (ou virtuais). Esses recursos de computação podem incluir rede, armazenamento, computação (CPU+memória). A virtualização é uma das principais técnicas usadas para realizar plataformas de computação em nuvem.

## Vantagens

* + Uma máquina virtual fornece a abstração de uma máquina para cada um dos usuários de uma máquina física.
  + Permite compartilhar os mesmos recursos físicos entre vários usuários.
  + Desacoplamento entre os recursos físicos e virtuais.

## VMM

O VMM fornece a ilusão de uma máquina idêntica à máquina original e controla o hardware. O software é executado em uma VM em completo isolamento de outras VMs, com apenas pequenas reduções na velocidade.

# Defenições

## Máquina Virtual (VM)

* É a simulação da “máquina real”, com um “hardware virtual” criado e gerenciado pelo VMM
* Pode haver várias instâncias de VM em execução, elas podem:
  + comportar-se como se fossem máquinas distintas, isoladas
  + executar seus aplicativos sem modificações e eficientemente

Virtual Machine Monitor (VMM)

* Fornece as abstrações de hardware virtual para a VM
* Um par (VM+VMM) constitui um “processo” de hypervisor

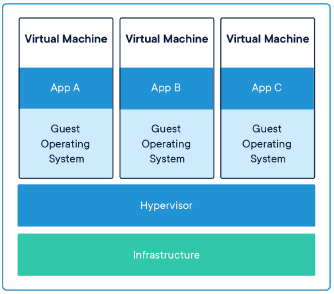
## Hypervisor

* É um SO executa os processos (VM/VMM)
* Às vezes, os termos VMM e hipervisor são usados para a mesma coisa

## Language VM

Fornece um ambiente com um conjunto de instruções específico e bibliotecas padrão para aceder os recursos do computador. Executa como um aplicativo em um SO e os programas são compilados para o código VM do idioma e executados no idioma VM.

# Virtualização - Implementação

A virtualização permite executar um SO dentro/em cima de outro SO

* Host OS/Hypervisor: rodando na máquina real
* SO convidado: corre dentro/sobre o SO host

O VMM (ou hypervisor) deve fornecer:

* **Segurança:** a VM não deve comprometer o VMM, que deve ter controle total dos recursos (reais e virtuais).
* **Fidelidade:** durante a execução, o comportamento do software na VM deve ser idêntico ao comportamento do hardware real.
* **Eficiência:** o VMM deve se interpor o mínimo possível enquanto o VM está executando seu código.

## Como o VMM se protege (de VMs)?

* O VMM pode manter o estado do processador, memória, IO e interpretar cada instrução da VM, atualizando o estado do processador/memória/IO de acordo. Isso é seguro, mas muito lentooo… Isso é chamado de emulador e é necessário executar programas compilados para um processador diferente.
* Para ser rápido, é necessário que o software da VM rode diretamente no processador, mas sem comprometer o sistema.
* O sistema operacional faz isso até certo ponto – um processo não pode acessar arbitrariamente os recursos do computador. Como?

1. Isolamento de memória entre processos;
2. As operações privilegiadas que permitiriam que um processo acessasse os recursos do computador só podem ser executadas no modo supervisor (modo S) – o código do SO é o único código que é executado no modo supervisor, garantindo que o SO possa controlar os recursos acessados por um processo .

Os VMMs podem usar a mesma abordagem?

* Não se houver apenas dois modos (modo S e de usuário)
* Se SO convidado da VM for executado em modo privilegiado, isso poderá comprometer o VMM, pois teria acesso a todas as instruções;
* Se o sistema operacional convidado da VM for executado no modo de usuário, ele não poderá ser executado corretamente, pois não poderá executar as instruções privilegiadas necessárias.
* O que é preciso?

1. O sistema operacional convidado da VM precisa pensar que está sendo executado no modo supervisor precisa ver o sinalizador S(supervisor);
2. O sistema operacional convidado da VM precisa ser capaz de emitir instruções privilegiadas, mas sua execução precisa ser controlada pelo VMM.

# Virtualização com Trap-and-emulate

Interceptação de instruções privilegiadas se executadas no modo de usuário

1. A VM executa o código do usuário diretamente na CPU (sem penalidade de desempenho)
2. O sistema operacional convidado é executado no modo de usuário (mas os sinalizadores vCPU indicam o modo S - portanto, o sistema operacional convidado pensa que está sendo executado no modo de usuário)
3. Quando o sistema operacional convidado executa uma instrução privilegiada, um trap altera a execução para VMM (em um trap, a execução do sistema operacional convidado é interrompida e uma função do VMM é chamada).
4. O VMM interpreta (emula) a instrução e retorna a execução para o sistema operacional convidado (no modo de usuário).

Todas as operações sensíveis (que se comportam de maneira diferente quando executadas nos modos U e S) são privilegiadas.

# Virtualização com Binary translation

Algumas instruções sensíveis não são privilegiadas, por exemplo, PUSHF, POPF.

* **Problema:** se o SO convidado for executado no modo U (com o sinalizador vCPU Smode), o efeito da execução da operação não será correto.
* **Ideia:** substituir a instrução sensitiva por um trap (forçando execução controlada no VMM).

Recorre à tradução binária (versões mais antigas do VMware)

* A VM executa o código do usuário diretamente na CPU (sem penalidade de desempenho).
* O sistema operacional convidado é executado no modo U (mas os sinalizadores vCPU indicam o modo S).
* O código binário do sistema operacional convidado é alimentado a um tradutor que emite código binário alterado para instruções confidenciais - não confidenciais e instruções privilegiadas não são modificadas.
* Instruções privilegiadas são executadas como trap-and-emulate.

## Desvantagens

Lentidão no código do sistema operacional convidado – necessidade de traduzir o código; várias instruções podem precisar ser capturadas em cada função do sistema operacional convidado;

# Virtualização com Paravirtualização

**Problema:** mesmo que a tradução de código possa ser rápida, ter várias instruções de cada função do SO convidado bloqueadas retarda a execução.

**Ideia:** Uma única trap seria necessária para chamar a função VMM. Além disso, a função VMM pode acessar os recursos diretamente (em vez de acessar uma versão virtualizada).

* O código-fonte do SO convidado é alterado para que as partes que usam instruções confidenciais não privilegiadas e privilegiadas sejam substituídas por chamadas de hipervisor que executam a mesma funcionalidade – por exemplo, uma função de sistema operacional convidado que acessaria o controlador de memória virtual virtualizada é substituída por uma chamada VMM/hypervisor que acessa o controlador de memória virtual diretamente;
* A VM executa o código do usuário diretamente na CPU (sem penalidade de desempenho).
* O SO convidado é executado no modo U (mas os sinalizadores vCPU indicam o modo S).
* As chamadas do hipervisor são implementadas como traps.

## Vantagens

Desempenho dentro de alguns por cento do caso não virtualizado.

## Desvantagens

Requer alterações no SO convidado: ok quando a fonte estiver disponível (por exemplo, Linux).

# Hoje em dia

A paravirtualização resolveu o problema de desempenho quando a tradução binária foi usada.

Não é necessário com o suporte de hardware para virtualização. Usado apenas nos poucos casos em que trap-and-emulate é muito alta para IO ou armazenamento.

# Virtualização de CPU’s

Virtualização de back-end

Técnicas de back-end executam a pilha do driver gráfico dentro da máquina virtual com o limite de virtualização entre a pilha e o hardware físico da GPU

#### GPU PCI passthrough technic

Permite uma VM para ter acesso direto à GPU por mapeamento das suas regiões de memória no espaço de endereço da VM.

* **Vantagem:** Desempenho quase nativo paraGPUs
* **Problemas:** Sem multiplexação.

Virtualização de front-end

A virtualização de front-end introduz um limite de virtualização em um nível relativamente alto na pilha e executa o driver gráfico no host/hipervisor

#### Technic

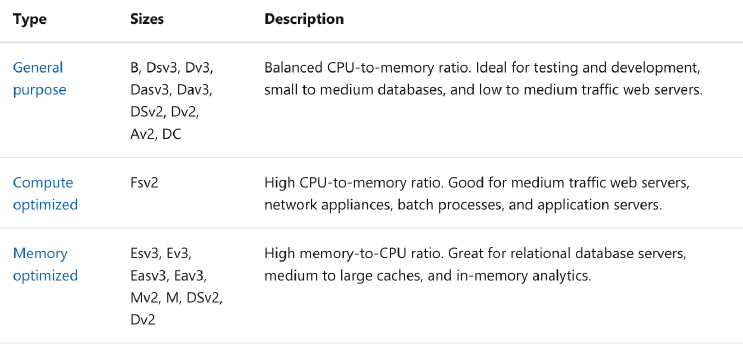
Permite que a GPU seja controlada pelo host/hipervisor e oferece suporte à multiplexação.

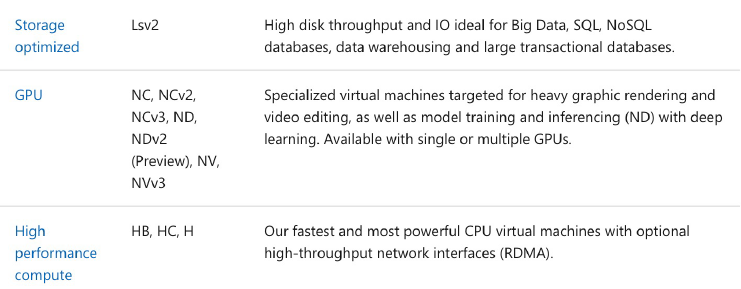
Na **emulação** de dispositivos, uma GPU virtual é emulada e a emulação sintetiza as operações gráficas do host em resposta às ações dos drivers do dispositivo convidado. Essas soluções permitem que a GPU seja executada em uma máquina diferente.

### IaaS – Azure VM

Suporta a criação de máquinas virtuais rodando Linux e Windows. Suporta a criação de VMs especialmente adaptadas para a execução de software específico (por exemplo, SAP). O utilizador seleciona as VMs que precisa.

# Type of each VM





# VM and faults

Unplanned Hardware Maintenance Event

Ocorre quando a plataforma Azure prevê que o hardware ou qualquer componente da plataforma associado a uma máquina física está prestes a falhar. Nesse caso, a plataforma emite um evento de manutenção de hardware não planejado e pode usar o **Live Migration** para migrar as MV’s do hardware com falha para uma máquina física saudável

#### Live Migration

Pausa a VM 🡪 migra memória, arquivos abertos e conexões de rede 🡪 retoma a VM.

A maioria dos sistemas inicia a migração de memória arquivos abertos e conexões de rede enquanto ainda está em execução, parando apenas por um período muito curto no final para migrar o que foi alterado entretanto

## An Unexpected Downtime

Ocorre quando o hardware ou a infraestrutura física da máquina virtual falha inesperadamente. Isso pode incluir falhas de rede local, falhas de disco local ou outras falhas no nível do rack. Quando detectadas, a cloud platform migram automaticamente a MV para uma máquina física saudável no mesmo datacenter.

Durante a recuperação, as MV’s sofrem um tempo de inatividade (reinicialização) e, em alguns casos, perda da unidade temporária. O SO anexado e os discos de dados são sempre preservados. As máquinas virtuais também podem sofrer inatividade no caso improvável de uma interrupção ou desastre que afete um datacenter inteiro ou até mesmo uma região inteira.

## Planned Maintenance events

São atualizações periódicas feitas na cloud platform fundamental para melhorar a confiabilidade, o desempenho e a segurança gerais da infraestrutura da plataforma.

As plataformas de nuvem tentam usar a manutenção de preservação de VM em todas as ocasiões possíveis, ou seja, para manter as VMs em execução (por exemplo, migrando-as para outras máquinas).

# Select the availability options

Cada **zona de disponibilidade** tem uma fonte de energia, rede e refrigeração distintas. Um aplicativo deve usar aplicativos e dados replicados em diferentes zonas para tolerar a perda de um datacenter.

#### Fault domains

É um grupo lógico de hardware fundamental que compartilha uma fonte de energia e um switch de rede comuns, semelhante a um rack em um datacenter local.

#### Update domains

É um grupo lógico de hardware fundamental que pode passar por manutenção ou ser reinicializado ao mesmo tempo.

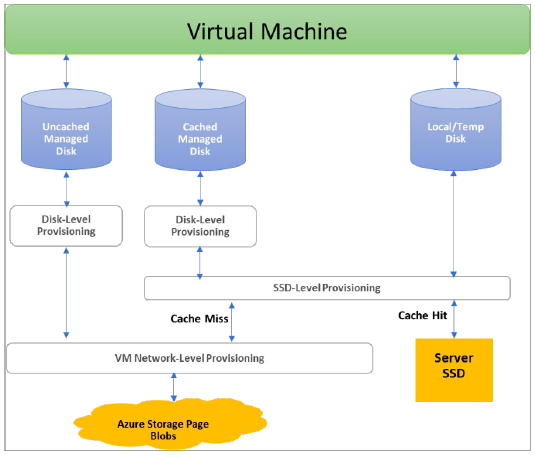
## Scale set

Os conjuntos de dimensionamento de máquinas virtuais do Azure permitem criar e gerenciar um grupo de VMs com balanceamento de carga. O número de instâncias de VM pode aumentar ou diminuir automaticamente em resposta à demanda ou a um cronograma definido. As máquinas virtuais em um conjunto de dimensionamento podem ser implantadas em várias regiões e domínios de falha para maximizar a disponibilidade e resiliência a interrupções.

## Availability set

É um agrupamento de VMs em um datacenter para fornecer redundância e disponibilidade. Em um conjunto de disponibilidade, as VMs são distribuídas automaticamente em diferentes domínios de falha e atualização. Essa abordagem limita o impacto de possíveis falhas físicas de hardware, interrupções de rede ou interrupções de energia.

# Disks and network options

**Managed disk** é um disco rígido virtual e é armazenado como blobs de páginas. São uma abstração sobre blobs de páginas, contêineres de blob e contas de armazenamento do Azure.

Os tipos de discos são:

* disco Ultra,
* unidade de estado sólido (SSD) Premium,
* SSD padrão e unidade de disco rígido (HDD) padrão.

O sistema provisiona por IOPS de disco e largura de banda. Além disso, provisiona a rede para acesso remoto. Localmente, o servidor implementa o provisionamento SSD para dados armazenados no SDD do servidor, incluindo disco com cache e discos locais e temporários.

# VM’s Vantagens

**Eficiência -** permite o uso eficiente de recursos e fornece isolamento.

**Flexibilidade -** recursos podem ser alocados conforme necessário.

**Restaurar e recuperar -** podem ser armazenadas como umúnico arquivo que pode ser copiado noutra fonte.

**Liberdade do sistema operacional -** Diferentes SO’s convidados podem existir no mesmo hypervisor.

**Desempenho e movimento -** Os hipervisores suportam a movimentação de uma máquina virtual de um host para outro em caso de degradação do desempenho na máquina host.

# VM’s Desvantagens

**Sobrecarga de desempenho -** Uma pilha de VM’s inclui: o SO convidado, o hipervisor e potencialmente o SO host

**Utilização eficiente de recursos -** O uso de vários SO’s no mesmo hipervisor duplica os recursos usados.

### Containers

Os contêineres fornecem:

* virtualização no nível do sistema operacional.
* namespace privado
* interface de rede e endereço IP
* sistemas de arquivos

Ao contrário das VMs, os contêineres compartilham o kernel do sistema host com outros contêineres. O código do aplicativo precisa usar a API do sistema operacional host.

## Promessas

Crie uma vez, execute em qualquer lugar

* Implantação mais rápida
* Portabilidade entre máquinas
* Controle de versão
* Gerenciamento simplificado de dependências

Nota: antes dos container existio o **chroot** que especificava uma dir como root, contudo o processo ainda pode acessar os dispositivos IO subjacentes, ele pode executar um segundo chroot se tiver privilégios suficientes. Todas as dependências do aplicativo precisam ser copiadas para o diretório chroot.

# Kernel namespaces

Os namespaces do kernel dividem os recursos do kernel (processos, usuários, pilhas de rede, etc.) em uma instância por namespace. Um processo visualiza apenas os recursos em seu namespace (existem atualmente 6 namespaces)

Os namespaces do kernel são usados para criar contêineres isolados que não têm visibilidade para objetos fora. Os processos em execução num contêiner compartilham o kernel subjacente com outros contêineres.

#### Implementação

System calls:

* **clone()** - cria um novo processo e um novo namespace (processo associado ao novo namespace)
* **unshare()** - cria um novo namespace e anexa o processo atual para ele.
* **setns()** - permite ingressar em um namespace existente.

#### Namespaces

**UTS Namespace -** nome de host e nome de domínio específicos do namespace.

**Network Namespace -** uma cópia da pilha de rede, com suas próprias rotas, regras de firewall e dispositivos de rede. Esta cópia da pilha de rede, com suas próprias rotas, regras de firewall e dispositivos de rede.

* Cada namespace de rede tem seus próprios endereços IP.
* Um dispositivo de rede pertence a exatamente um namespace de rede.
* Um soquete pertence a exatamente um namespace de rede.
* Comunicação entre dois namespaces de rede:
* Veth (ethernet virtual) é usado como um canal entre dois namespaces
* As tomadas também funcionam

**Mount Namespace -** Na criação, a árvore do sistema de arquivos é copiada para um novo espaço, com todas as montagens anteriores visíveis. Futuras montagens/desmontagens invisíveis para o resto do sistema.

**PID Namespace -** Processos em diferentes namespaces PID podem ter o mesmo ID de processo.

**User Namespace -** Um processo terá um conjunto distinto de UIDs, GIDs e capacidades

**IPC Namespace -** Cada namespace obtém seus próprios objetos IPC e filas de mensagens POSIX.

# Cgroups

Cgroups são um mecanismo para aplicar limites de recursos de hardware e controles de acesso a um processo ou coleção de processos, e os subsistemas relacionados fornecem um mecanismo hierárquico baseado em árvore, herdável e opcionalmente aninhado de controle de recursos.

Cgroups são usados para limitar o consumo de memória e CPU dos contêineres. Um contêiner pode ser redimensionado simplesmente alterando os limites de seu cgroup correspondente.

#### Implementação

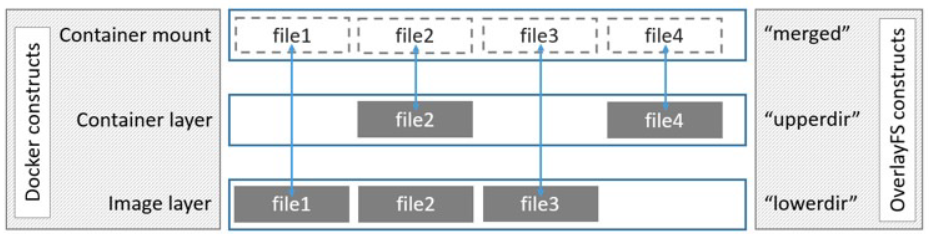
A implementação de cgroups requer alguns ganchos simples no restante do kernel: na fase de inicialização, processo de criação e destruição. Todas as operações em cgroups são executadas usando operações em um VFS (sistema de arquivo virtual)

# Copy-on-write File system

UnionFS é um sistema de arquivos copy-on-write que é a união de sistemas de arquivos existentes. Fornece uma visão unificada do sistema de arquivos, que combina todos os sistemas de arquivos empilhados.

Ao gravar no UnionFS, os dados substituídos são salvos em um novo caminho, específico desse contêiner.

* Assim, as gravações de um contêiner não afetam as leituras de outro contêiner



O UnionFS permite que vários contêineres compartilhem dados comuns. Cada camada é armazenada apenas uma vez.

# Docker

Construído sobre namespaces do kernel, cgroups, unionFS e capacidades. Cada contêiner obtém seu próprio conjunto de namespaces e Cgroups.

* Os namespaces isolam os contêineres uns dos outros: um contêiner não pode nem ver a lista de processos em outro contêiner.
* Cgroups permitem que o administrador isole os recursos usados por cada contêiner e seus filhos.
* A execução do daemon docker requer privilégios de root.
* O Docker fornece uma lista de permissões de recursos para usuários root dentro de um contêiner.

## Docker engine

**Daemon Docker** (dockerd) gerencia objetos Docker como imagens, contêineres, redes e volumes. O cliente docker envia solicitações ao daemon docker.

Um **registro do Docker** armazena imagens do Docker.

Uma **imagem** é um modelo somente leitura com instruções para criar um contêiner Docker. Frequentemente, uma imagem é baseada em outra imagem, com alguma customização adicional. Uma imagem do Docker pode ser criada a partir da especificação em um Dockerfile.

## Docker networking

A **Bridge network** permite conectar vários contêineres docker em execução no mesmo host docker. Uma rede pode ser criada usando o docker network create. O driver de rede de sobreposição cria uma rede distribuída entre vários hosts de daemon do Docker

## Docker Storage

Por padrão, todos os arquivos criados dentro de um contêiner são armazenados em uma camada de um contêiner que possa ser escrito e não persistente. O Docker tem duas opções para contêineres armazenarem arquivos na máquina host, para que os arquivos sejam persistidos mesmo após o container parar: volumes e montagens de ligação.



Os **Volumes** são armazenados em uma parte do sistema de arquivos host que é gerenciado pelo Docker. Os processos não Docker não devem modificar esta parte do sistema de arquivos. Os volumes são a melhor maneira de persistir dados no Docker.

As **Bind mounts** podem ser armazenadas em qualquer lugar no sistema host. Eles podem até ser arquivos ou diretórios importantes do sistema. Os processos não Docker no host Docker ou em um contêiner Docker podem modificá-los a qualquer momento.

As **tmpfs mounts** são armazenadas apenas na memória do sistema host e nunca são gravadas no sistema de arquivos do sistema host.

## Docker compose

O Docker Compose permite definir e executar aplicativos Docker de vários contêineres.

# Docker Swarm

O Swarm é um cluster de instâncias do Docker, e consiste em vários hosts do Docker que são executados no modo Swarm e atuam como managers e trabalhadores.

Ao criar um serviço, você define seu estado ideal (número de réplicas, rede e recursos de armazenamento disponíveis para ele, portas que o serviço expõe ao mundo externo e muito mais).

O Docker trabalha para manter esse estado desejado. Por exemplo, se um nó de trabalho ficar indisponível, o Docker agendará as tarefas desse nó em outros nós.

## Nodes

Um nó é uma instância do mecanismo do Docker que participa do enxame. Normalmente, consiste em nós do Docker distribuídos em várias máquinas físicas e em nuvem. O nó gerenciador recebe definições de serviço e despacha unidades de trabalho chamadas de tarefas para nós trabalhadores Os nós gerenciadores também executam as funções de orquestração e gerenciamento de cluster necessárias para manter o estado de enxame. Os nós do trabalhador recebem e executam tarefas despachadas dos nós do gerenciador.

## Service

Um serviço é a definição das tarefas a serem executadas nos nós do gerenciador ou do trabalhador. Um serviço é a raiz primária da interação do usuário com o enxame.

Um serviço especifica qual imagem de contêiner usar e quais comandos executar. No modelo de serviços replicados, o gerente do enxame distribui um número específico de tarefas de réplica entre os nós com base na escala definida no estado desejado.

Para serviços globais, o enxame executa uma tarefa para o serviço em cada nó disponível no cluster.

## Load Balancing

O gerenciador de swarm usa balanceamento de carga de entrada para expor os serviços externamente ao swarm. O swarm manager pode atribuir automaticamente um PublishedPort ao serviço ou você pode configurar um PublishedPort para o serviço. Componentes externos, como balanceadores de carga em nuvem, podem acessar o serviço no PublishedPort de qualquer nó no cluster. Todos os nós no enxame roteiam conexões de entrada para uma instância de tarefa em execução. O modo Swarm atribui automaticamente a cada serviço no Swarm uma entrada DNS. O gerenciador de swarm usa balanceamento de carga interno para distribuir solicitações entre serviços dentro do cluster com base no nome DNS do serviço.

## Swarm services vs standalone container

É possível modificar a configuração de um serviço, incluindo as redes e volumes aos quais está conectado, sem a necessidade de reiniciar manualmente o serviço. O Docker atualizará a configuração, interromperá as tarefas de serviço com a configuração desatualizada e criará novas que correspondam à configuração desejada.

# Kuebernetes

O Kubernetes é uma plataforma de código aberto para automatizar a implantação, dimensionamento e operações de contêineres de aplicativos.

* Horizontal Scalability
* Self-healing
* Service Discovery
* Automated Rollbacks

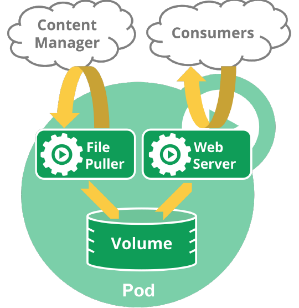
## Objects

**Pod**: encapsula o contêiner de um aplicativo (ou vários contêineres), recursos de armazenamento, um IP de rede exclusivo e opções que controlam como o(s) contêiner(es) deve(m) ser executado(s).

**Service**: um serviço é uma abstração que define um conjunto lógico de pods e uma política para acessá-los.

**Volume**: um volume é um diretório acessível aos Containers em um Pod. Um volume do Kubernetes tem o mesmo tempo de vida do pod que o inclui.

**Namespace**: Namespaces fornecem um escopo para nomes. Os nomes dos recursos precisam ser exclusivos dentro de um namespace, mas não entre namespaces.



Init containers

Eles são executados e concluídos antes que os contêineres do aplicativo sejam iniciados. Se um init contêiner falhar, o pod será reinicializado. Isso pode ser usado para fazer inicializações antes de iniciar o serviço principal.

## Pod resources

#### Networking

Cada pod recebe um endereço IP exclusivo. Cada contêiner em um pod compartilha o namespace da rede, incluindo o endereço IP e as portas de rede. Os contêineres dentro de um pod se comunicam usando localhost. Quando os contêineres em um pod se comunicam com entidades fora do pod, eles devem coordenar como usam os recursos de rede compartilhados (como portas).

#### Storage

Um pod pode especificar um conjunto de volumes de armazenamento compartilhado. Todos os contêineres no pod podem acessar os volumes compartilhados, permitindo que esses contêineres compartilhem dados. Os volumes também permitem que os dados persistentes em um pod sobrevivam caso um dos contêineres precise ser reiniciado.

## Services

Um serviço é uma abstração que define um conjunto lógico de pods e uma política para acessá-los. Por exemplo, considere um back-end de processamento de imagem sem estado que está sendo executado com 3 réplicas. Os frontends não se importam com qual backend eles usam e se eles mudam.

As APIs do Kubernetes para descoberta de serviço permitem consultar Endpoints, que são atualizados sempre que o conjunto de pods em um serviço muda. Para aplicativos não nativos, o Kubernetes oferece maneiras de colocar uma porta de rede ou um balanceador de carga entre seu aplicativo e os pods de back-end.

#### Tipos de serviços

**ClusterIP:** expõe o serviço em um IP interno do cluster. Este é o ServiceType padrão.

**NodePort:** Expõe o Serviço no IP de cada Node em uma porta estática (o NodePort). Um Serviço ClusterIP, ao qual as rotas do serviço NodePort são criadas automaticamente.

**LoadBalancer:** expõe o serviço externamente usando o balanceador de carga de um provedor de nuvem. Os serviços NodePort e ClusterIP, para os quais o balanceador de carga externo roteia, são criados automaticamente.

**ExternalName**: mapeia o serviço para o conteúdo do campo externalName (por exemplo, foo.bar.example.com).

## Volumes

Os arquivos em disco em um contêiner são efêmeros. Isso leva a problemas para aplicativos que não são sem estado.

* Primeiro, quando um Container falha, ele será reiniciado, mas os arquivos serão perdidos - o Container inicia com um estado limpo.
* Em segundo lugar, ao executar contêineres juntos em um pod, muitas vezes é necessário compartilhar arquivos entre esses contêineres.
* Um volume é um diretório acessível aos contêineres em um pod.
* Um volume do Kubernetes tem o mesmo tempo de vida do pod que o inclui. Consequentemente, um volume sobrevive a qualquer contêiner executado no pod e os dados são preservados nas reinicializações do contêiner.
* Quando um Pod deixa de existir, o volume também deixa de existir. O Kubernetes oferece suporte a muitos tipos de volumes, e um pod pode usar qualquer número deles simultaneamente. Alguns volumes são específicos para uma determinada nuvem e alguns são persistentes (no sentido de que delineiam um Pod).

## Namespace

O Kubernetes suporta vários clusters virtuais apoiados pelo mesmo cluster físico. Esses clusters virtuais são chamados de namespaces.

Os namespaces destinam-se ao uso em ambientes com muitos usuários espalhados por várias equipes ou projetos.

Namespaces fornecem um escopo para nomes. Os nomes dos recursos precisam ser exclusivos dentro de um namespace, mas não entre namespaces.

Os namespaces não podem ser aninhados uns dentro dos outros e cada recurso do Kubernetes só pode estar em um namespace.

## Kubernetes controllers

**Deployments**: uma implantação fornece atualizações declarativas para Pods e ReplicaSets.

**ReplicaSet**: o objetivo de um ReplicaSet é manter um conjunto estável de réplicas de pods em execução a qualquer momento.

**DaemonSet**: Um DaemonSet garante que todos (ou alguns) nós executem uma cópia de um pod. À medida que os nós são adicionados/removidos ao cluster, os pods são adicionados/excluídos deles.

**StatefulSet**: StatefulSet é o objeto de API de carga de trabalho usado para gerenciar aplicativos com estado.

**Job**: um trabalho cria um ou mais pods e garante que um número especificado deles seja encerrado com sucesso

## Deployment

A Deployment provides declarative updates for Pods and ReplicaSets. A Deployment describes the desired state of the system. The Deployment Controller changes the actual state to the desired state at a controlled rate.

## ReplicaSet

Um ReplicaSet garante que um número especificado de réplicas de pod esteja em execução a qualquer momento. Um deployment gerencia ReplicaSets.

## DeamonSet