Resumos ASCN

Eduardo Silva, Ivo Baixo e José Carlos Magalhães,
com o apoio do incrível prp[1]

January 2022

[1]succ jno

Conteúdo

1 Distributed Apps

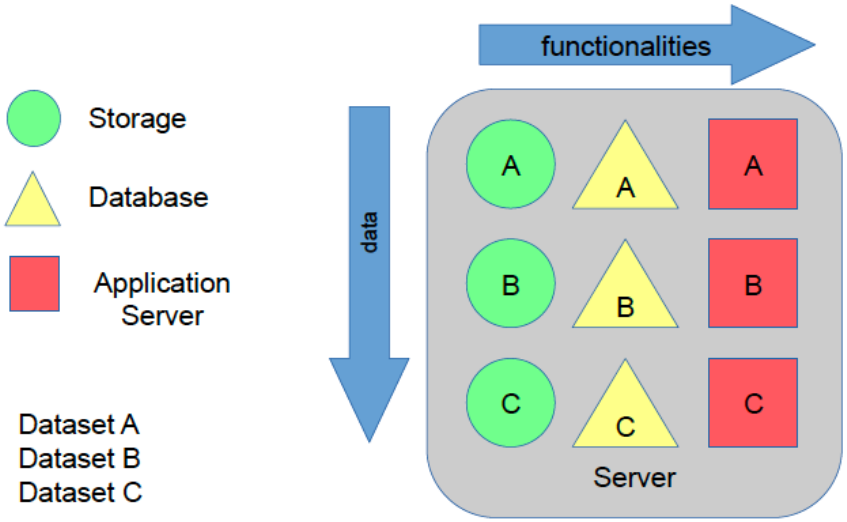
1.1 Principais preocupações

Porquê sistemas distribuídos?

- Modularidade ou com a desacoplação de diferentes preocupações;
- Desempenho;
- Fiabilidade.

1.2 Principais arquiteturas

1.2.1 Sistemas monolíticos



Uma arquitetura monolítica é um modelo unificado tradicional para o projeto de um programa de software. Monolítico, neste contexto, significa que é composto todo numa só peça, ou seja, **possui vários serviços para vários destinos no mesmo servidor**.

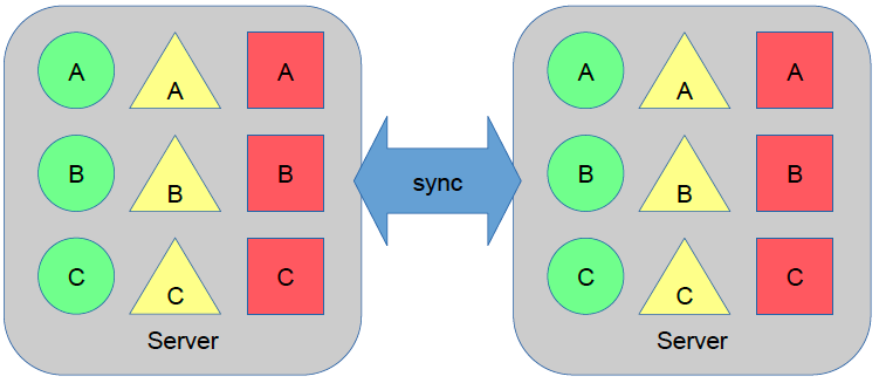
1.2.2 Sistema distribuído

Principais preocupações:

- Replicação;
- Particionamento;
- Orientação a serviços.

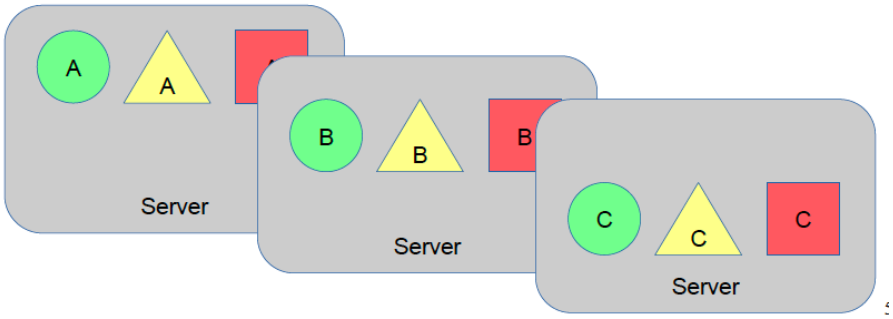
Todos os endereços dimensionam um serviço/aplicação e não são mutuamente exclusivos, ou seja, podem ser combinados.

1.2.3 Replicação



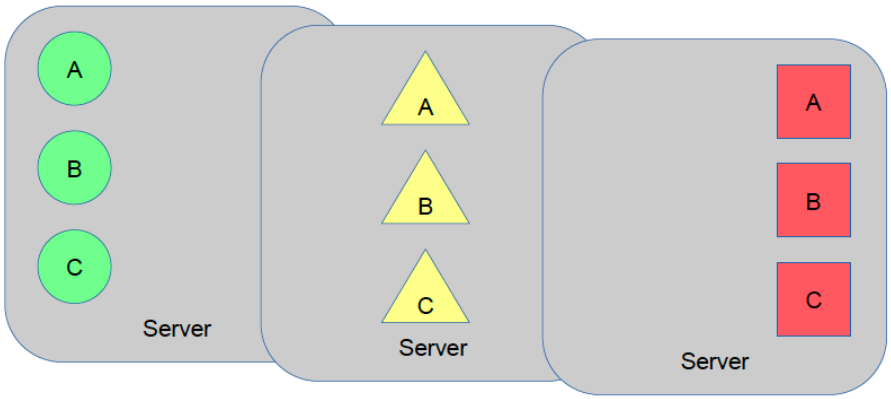
Várias cópias dos mesmos dados e das mesmas funcionalidades. Aborda **resiliência e a expansão** (*scale-out*).

1.2.4 Partição



O servidor é dividido horizontalmente (*Sharding*), há uma expansão dos endereços e pode ser aplicado à computação, dados, etc...

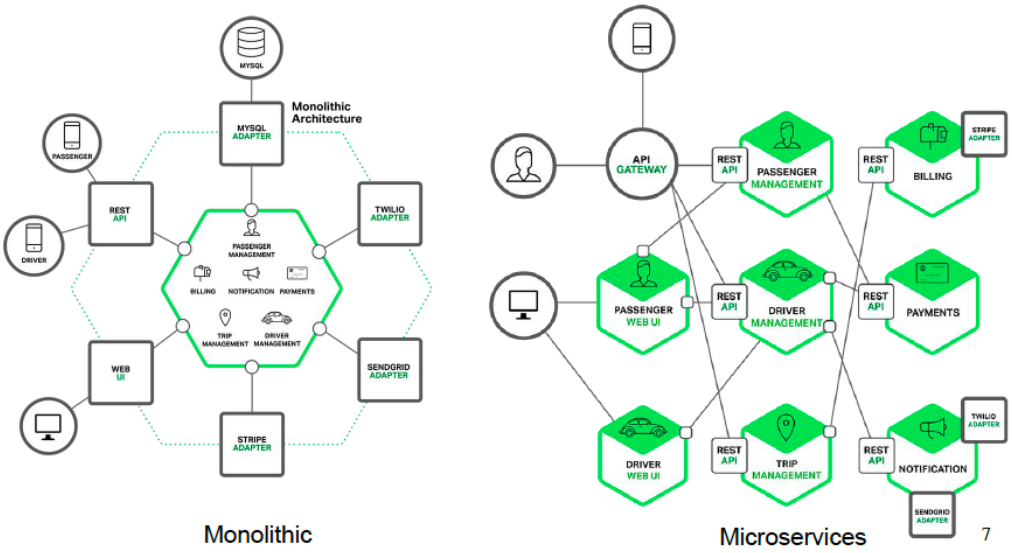
1.2.5 Arquitetura Orientada a Serviços (SOA)



O servidor é dividido verticalmente. Aborda a expansão e a modularidade.

Exemplo: microserviços

1.3 Monolithic to Microservices

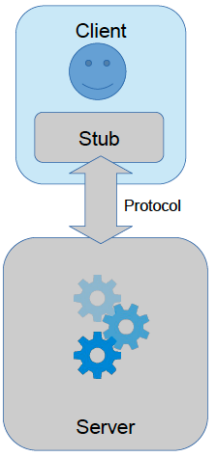


1.3.1 Microserviços

- Cada serviço implementa uma funcionalidade específica;
- Os serviços podem ser dimensionados de forma independente;
- Consistência;
- A implementação e os testes são complexos;

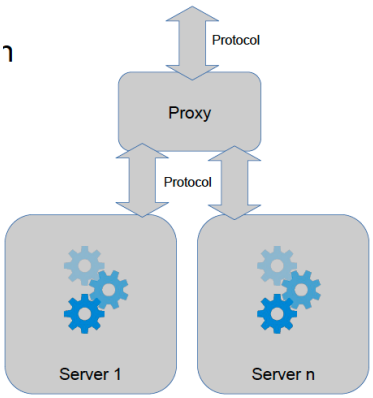
1.4 Client - Server

- As funcionalidades e os dados estão no servidor;
- O cliente possui um stub que usa para comunicar com o servidor e o stub faz parte do pacote de software do servidor.



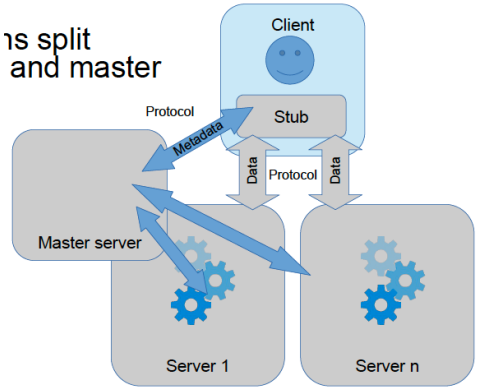
1.5 Proxy Server

- Pode encapsular N servidores, uma vez que o cliente apenas comunica com o proxy;
- O Proxy é um bottleneck de performance e disponibilidade.



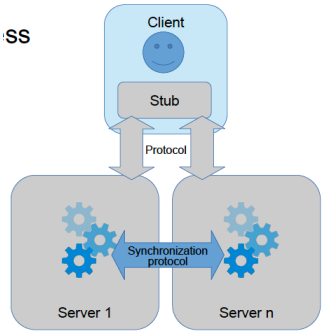
1.6 Master - Server

- As funções de um proxy são divididas entre o stub do cliente e o servidor master.
- O cliente faz parte da comunicação com os servidores e outra com o Master que, por sua vez, comunica também com os servidores.



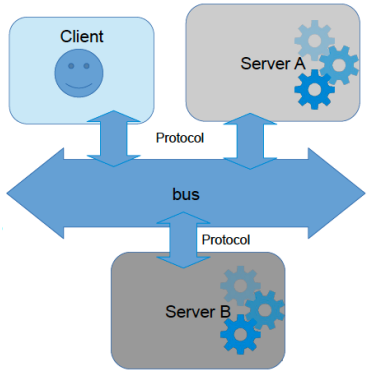
1.7 Server group

- Todos os servidores podem processar pedidos;
- É necessário um sistema de coordenação entre os servidores - *Synchronization Protocol*.



1.8 Bus

- Existe um *bus* comum a todos, que encaminha as mensagens;
- Os participantes publicam e consomem mensagens do *bus*;
- Há a separação dos produtores e dos consumidores.



1.9 Multi-tier

- Cada servidor age como um cliente do próximo tier.
- Permite o desenvolvimento independente e um dimensionamento de diferentes funcionalidades.

1.10 State in multi-tier

- Cada camada tem um estado persistente,é mais difícil de replicar e fazer shard do estado;
- Computação mais fácil de replicar e fazer shard.
- Upper tiers (Web browser): Sem estado ;
- Middle tiers (Application Server) : estado Transente e cached ;
- Lower Tiers (Database) : estado Persistente.

2 Provisioning

Provisioning: A ação de fornecer algo para uso, por exemplo: Server provisioning, Storage provisioning, Network provisioning, VM provisioning, User provisioning.

Deployment é o processo de instalar ou atualizar uma aplicação ou serviço numa máquina.

Ansible:

- Inventário:
  - Agrupamento de máquinas a dar deploy (hosts).
- Módulo:
  - Unidade de trabalho/código reutilizável que pode ser corrida no terminal ou no playbook.
- Task:
  - Combinação de um módulo e os agrupamentos a este passados, para criar uma ação.
- Playbook:
  - Conjunto de tasks.
- Handler:
  - Tipo de task especial que responde a notificações.
- Role:
  - Componente reutilizável que encapsula variáveis, tasks e handlers.

Vault: Permite manter dados confidenciais, como senhas ou chaves em arquivos criptografados, em vez de ter como texto simples em cartilhas ou papéis.

Inventário Dinamico:

- Quando a provisão aocntece dinamicamente os endereços são desconhecidos, este problema tem duas soluções: Manualmente (ir à consola e procurar os endereços) e automático(usar o inventário dinamico).
- Ansible fornece o gcp compute plugin: Queries GCP, permite filtros, Enables dynamic grouping

3 Cloud Services

Os serviços de cloud podem ser divididos em 3 níveis de abstração:

- Infrastructure-as-a-Service (IaaS);
- Platform-as-a-Service (PaaS);
- Software-as-a-Service (SaaS).

3.1 Infrastructure-as-a-Service (IaaS):

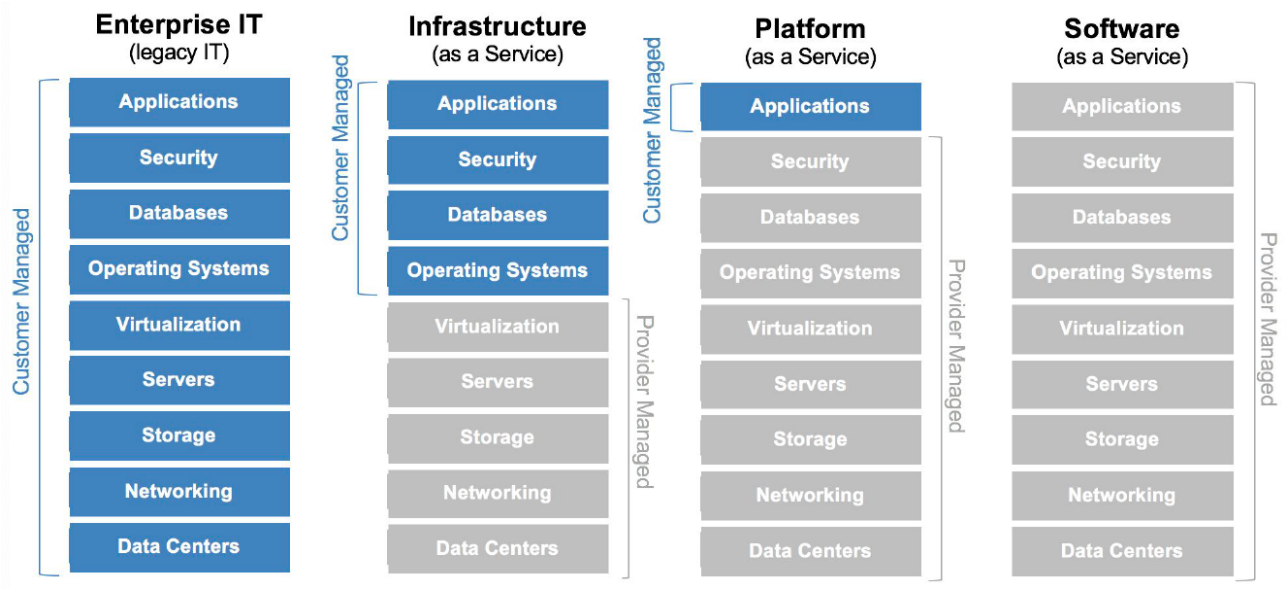
- Fornece recursos de hardware virtualizados, como computação, armazenamento e rede;
- Os recursos são alocados a pedido e de forma pay-per-use;
- Exemplos: Amazon EC2, Google Compute Engine.

3.2 Platform-as-a-Service (PaaS):

- Oferece um encapsulamento de uma abstração do ambiente de desenvolvimento que pode ser usada para desenvolver, fazer deploy ou executar aplicações.

3.3 Software-as-a-Service (SaaS):

- Apresenta aplicações completas ou software genérico como bases de dados;
- Oferecido como um serviço e acessível como um serviço web ou através de um browser.
- Exemplos: GMAIL



3.4 OpenStack

- Fonte software aberta para criar nuvens privadas/publicas;
- Controla grandes "pools"de computação,sotrage e networking recursos por todo o dataCenter;
- Gerido por todo o Dashboard ou via OpenStack API.

3.4.1 Cinder

- Block storage;
- virtualiza a gerenciamento de dispostivos de armazenamento em bloco
- fornece end users com self service API para solicitar e consumir os respetivos recursos sem exigir qualquer conhecimento sobre onde estes possam estar guardados.

3.4.2 Swift

- Altamente disponivel, distribuido, e eventualmente objeto consistente;
- Ideal para guardar data que não estão estruturados(imagens Vm, videos)

3.4.3 Ephemeral Storage vs Cinder vs Swift

3.5 De IaaS para PaaS

- De alocação gerida e provisão de recursos para infraestrutura gerida;
- Recursos atuais tornam-se transparentes;
- Foca-se na aplicação;
- O user pode-se focar na funcionalidade de fazer deploy em vez de que recursos que são necessários.

3.6 De PaaS para SaaS

- Serviços específicos são fornecidos
- Não há deploument item - A DB (suponho que seja Data BAse) é exposta a um cliente e usado como tradicional DB com a minima configuração necessária e acesso remoto.

3.7 Sumário- IaaS, PaaS, SaaS

3.7.1 Vantagens:

- Conveniência:
  - De IaaS:
    - evita custos adiantados na gestão da infraestrutura e hardware
    - fácil de fazer deploy de aplicações
  - Para PaaS:
    - focado no desenvolvimento da aplicação e seus requerimentos
    - ferramentas já prontas a utilizar
  - Para SaaS:
    - vantagem em existir componentes

- Elasticidade:
  - De IaaS:
    - Ilusão de recursos virtuais infinitos
    - aumenta e diminui recursos de acordo com o que for necessário (storage, espaço etc)
  - Para PaaS e SaaS:
    - Não há necessidade de gerir manualmente a elasticidade
  - Para Saas:
    - rápida interação em diferentes soluções de cloud
- Velocidade:
  - De IaaS:
    - Infraestrutura já está instalada e configurada
  - Para PaaS:
    - Framework já está instalada e configurada
  - Para Saas:
    - rápida interação em diferentes soluções de cloud

3.7.2 Desvantagens:

- Perda de Controlo:
  - De IaaS:
    - Sem controlo sobre especifico hardware e virtualização de software
    - sem possibilidade de optimizar a infraestrutura
  - Para PaaS:
    - Sem controlo Sobre especifico hardware e plataforma PaaS;
  - Para SaaS:
    - third-party cloud applications
- Segurança:
  - IaaS, PaaS e SaaS:
    - Tão seguro quanto o fornecedor
    - soluções para tem que vir do fornecedor
    - Privacidade do data em infraestruturas third-party.

3.8 Segurança

Em termos de segurança, estes 3 tipos de serviços são tão seguros quanto os provider, ou seja, qualquer vulnerabilidade no provider, é uma vulnerabilidade na aplicação.

4 Data Centers

4.1 Arquitetura e segurança

- Fornece um local físico seguro para servidores, armazenamento e equipamentos de rede;
- Área para equipamentos e espaço livre de 50/50;
- Todos os pontos de entrada e portas devem ser controlados por dispositivos de controle de acesso e/ou pessoas que monitorizam e restringem o acesso das pessoas que entrem no data center;
- A escolha da elevação do piso é feita a partir de questões como a localização dos equipamentos, a gestão dos cabos (energia, rede ou refrigeração) assim como o fluxo de ar frio.

4.2 Outros cuidados

- É necessário ter UPS que protejam contra falhas de energia, ou geradores;
- Sistemas contra a prevenção de incêndios, que não utilizem água;
- Sistemas que permitam que os equipamentos corram sem aquecerem, assim como manter uma baixa humidade no data center. Este sistema de climatização, deve ser redundante de forma a lidar com falhas ou manutenção;
- Necessita de uma largura de banda, para o armazenamento dos dados e a correta gestão dos mesmos, e de conexões ISPs redundantes. Cabos Cat 6/7 de cobre, e fibra em multimodo ou monomodo.

5 Virtualização

Virtualização é uma técnica que permite criar um dispositivo ou recurso virtual baseado em software que, na prática, é uma abstração fornecida por recursos de hardware ou software já existentes.

5.1 Vantagens

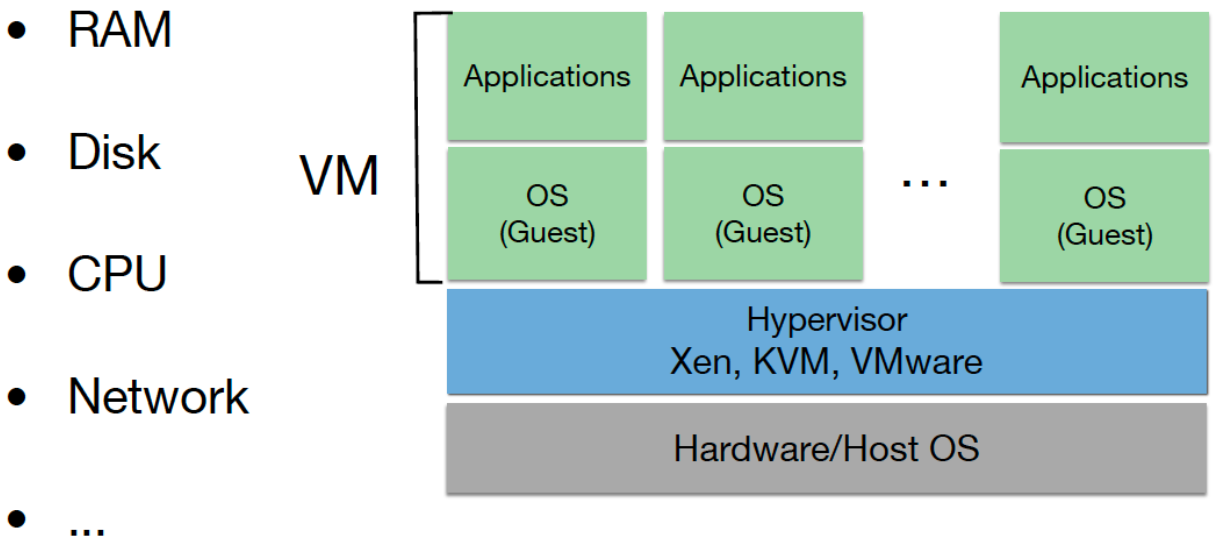
- Heterogeneidade:
  - Recursos virtuais podem ser fornecidos em cima de diferentes recursos físicos;
  - Um recurso virtual pode suportar diferentes aplicativos ou sistemas operativos, enquanto recorre **ao mesmo** hardware físico (VM's, por exemplo).
- Transparência:
  - A interação de um utilizador com um recurso virtual é semelhante à interação com o recurso físico.
- Isolamento (Isolation):
  - Recursos virtuais são isolados do hardware/software subjacente em termos de:
    - Segurança;
    - Performance / desempenho;
    - Casos de falhas (mesmo quando os dados ou o OS são corrompidos).
- Otimização de recursos:
  - Recursos físicos podem ser utilizados para suportar mais aplicações ou clientes, ou seja há:
    - Consolidação de servidores;
    - Custos mais baixos.
- Gestão simplificada:
  - Um recurso virtual é mais simples de gerir, por exemplo, é fácil de fazer migração de dados, assim como o backup de uma máquina virtual.

5.2 Desvantagens

- Performance:
  - A virtualização de recursos, por norma, incluí uma penalização de performance, por exemplo quando acontece **overprovisioning**.
    - Deployment de mais recursos virtuais do que os fisicamente disponíveis.
- Segurança:
  - Se o isolamento não for tratado de uma forma adequada, ou se um user/admin malicioso tiver acesso aos recursos físicos (por exemplo ao servidor), a segurança poderá ser comprometida;
- Dependability / Confiabilidade:
  - A falha do recurso físico pode resultar na falha de vários recursos virtuais.

5.3 Máquinas Virtuais

- Permitem a execução de diferentes sistemas operativos no mesmo servidor físico;
- Permitem a consolidação de recursos do servidor;
- As operações são intercaladas por um **Hypervisor** e são executadas no hardware físico. É este **Hypervisor** que controla as interações entre as VMs e SO/hardware host, tendo acesso ao disco partilhado, à rede, CPU e RAM.



5.3.1 CPU

- Corre um *time slicing:* os requests de processamento são divididos e compartilhados entre as VMs;
- Semelhante à execução de vários processos num host físico;
- Overcommitting vCPUs (virtual CPUs) pode levar a um mau desempenho.

5.3.2 Ram e Storage

- Cada VM aloca uma quantidade específica de RAM e Storage;
- O sistema de armazenamento deve lidar com várias leituras / escritas de forma eficiente;
- Com *Thin-provisioning*, os recursos de armazenamento são alocados conforme necessário.

5.3.3 Network

- As VMs partilham largura de banda da rede;
- Cada VM pode ser configurada de forma diferente ao nível da rede:
  - **Host-Only**: partilha o namespace de rede do host, e a VM só tem acesso ao host;
  - **Nat**: Faz com que a atividade de rede fosse feita pelo host (identidade de rede única) e a VM tem acesso a recursos externos;
  - **Bridge**: Usa o hypervisor para atribuir um IP específico à VM e a VM é vista como outro nó na rede física.

5.4 Modos de virtualização

5.4.1 Full Virtualization

- O OS da VM não é modificado pois o hypervisor simula todos os dispositivos de hardware no ambiente virtual.
- Vantagens:
  - Como não existem modificações no SO guest, existe uma maior variedade de tipos de OS suportados e, por isso, a migração/portabilidade de VMs é mais fácil.
- Desvantagens:
  - Todas as operações têm de ser processadas por duas camadas (VM e hypervisor), que faz com que exista uma performance pior de CPU e IO.

5.4.2 Paravirtualization

- Requer alterações no OS da VM guest, o que é pior em termos de manutenção e portabilidade;
- Operações custosas são efetuadas diretamente no host nativo e, portanto, a performance é melhor.

5.5 Tipos de virtualização

5.5.1 Tipo 1 - Bare Metal Hypervisor

Neste tipo, o hypervisor é o próprio sistema operativo, e a performance é superior.

5.5.2 Tipo 2 - Hosted Hypervisor

O hypervisor é um componente de software que corre em cima de um OS genérico. Neste caso, a portabilidade é superior.

Link úteis:

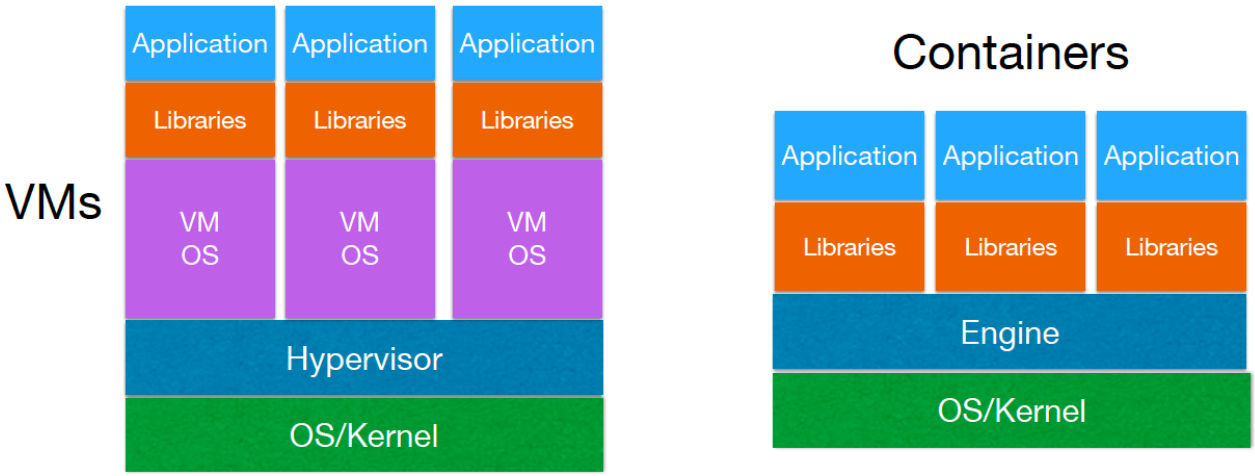
- Sobre Virtualização: `https://www.youtube.com/watch?v=FZR0rG3HKIk`
- Sobre Full irtualization: `https://www.youtube.com/watch?v=CLR0pq9dy4g`

- Sobre Paravirtualization: `https://www.youtube.com/watch?v=1PYNcmZTjiE`

5.6 Containers

- Executam versões diferentes e isoladas do mesmo software ou aplicação (por exemplo base de dados), num ambiente partilhado com o OS/Kernel;
- Portabilidade/migração entre os servidores;
- Fácil de "empacotar" o software, aplicações e as suas dependências.
- São ambientes virtuais leves que isolam um grupo de processos e recursos (CPU, memória, disco, etc);
- Corre no OS do host.

5.6.1 Container vs VM



VMs e Container têm objetivos diferentes:

- VMs são mais úteis quando precisamos de virtualizar o servidor inteiro;
- Containers são úteis para gerir várias livrarias e aplicações no host.

Vantagens de um Container:

- Mais rápido para fazer testes, provisionamento e migração;
- Melhor utilização dos recursos (mais leve) e por isso, com melhor desempenho;
- Pode ser "lançado" tanto em servidores físicos como virtualizados.

5.6.2 Vantagens de uma VM:

- Melhor isolamento / segurança, dado que o OS não é partilhado;
- Maior flexibilidade para correr OS diferentes.

Link úteis:

- `https://www.youtube.com/watch?v=cjXI-yxqGTI`

6 Storage

6.1 Relevância de sistemas de armazenamento

- "Pedra angular" para infraestruturase sistemas de gestão de dados;
- O desempenho é fundamental pois armazenamentos e recuperação de dados lentos, traduzem-se em aplicações lentas;
- Persistência e disponbilidade de dados.

6.2 Tipos de armazenamento

6.2.1 Archival

- Dados de gravação única (normalmente);
- A taxa de transferência (throughtput) é fornecida em relação à latência;
- Sequential workloads;
- Grandes quantidades de dados devem ser gravados/lidos de forma eficiente.

6.2.2 Backup

- Dados atualizados esporadicamente;
- A taxa de transferência ainda é favorecida em relação à latência;
- Sequential workloads (principalmente);
- As atualizações de dados devem ser consideradas.

6.2.3 Primary Storage

- Fornece suporte de armazenamento para base de dados, análises, etc...
- Alta taxa de transferência e baixa latência são desejáveis;
- Sequential e Random workloads;
- Os dados são atualizados com frequência

7 Monitoring

Um monitor é uma entidade que **observa** a atividade de um sistema.

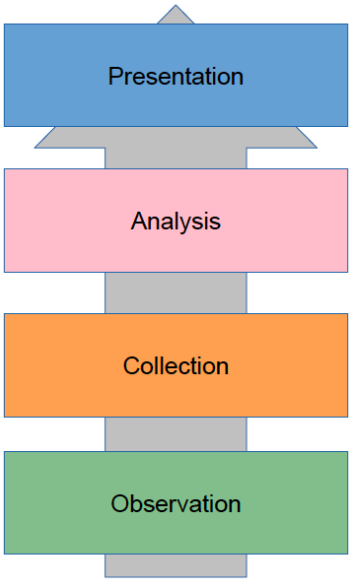
7.1 Conceitos

- **Domínio:** conjunto de atividades observadas;
- O detalhe varia:
  - No tempo, de acordo com o *input rate*;
  - No cenário, de acordo com a resolução.
- Impõe um **overhead** no sistema, mudando assim a atividade observada.

7.2 Classificação

- O que "desencadeia" (triggers) a observação: **Event-driven** vs **Sampling**;
- Quando é que a observação está disponível: **Online** vs **Batch**;
- O que é observado: **Hardware** vs **Software**;
- **Centralized** vs **Distributed**.

7.3 Arquitetura do monitor



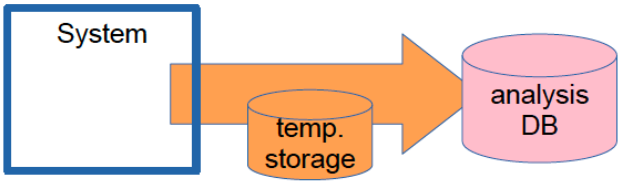
- **Presentation:** produz relatórios, alarmes, etc;
- **Analysis:** filtra, relaciona e resume dados;
- **Collecion:** coleciona e normaliza dados;
- **Observation:** observação de eventos "raw" do sistema.

7.3.1 Presentation

- Observação passiva ou espionagem (spying):
  - Network sniffer.
- Instrumentation:
  - Contadores embutidos no sistema (hardware e SO);
  - Geração de logs.
- Sonda com pedidos adicionais:
  - Ping.

7.3.2 Collection

- Push Data vs Pull Data:
  - Dependa da configuração, legacy systems, etc.
- Confiabilidade e persistência:



- Sincronização de tempo em sistemas distribuídos;
- Exemplos: Collectd, logstash.

7.3.3 Analysis

- Tarefa que lida com o processamento de dados:
  - Série temporal;
  - Procura.
- "Big Data" em grandes infraestruturas.
- Exemplos: ElasticSearch, Graphite.

7.3.4 Presentation

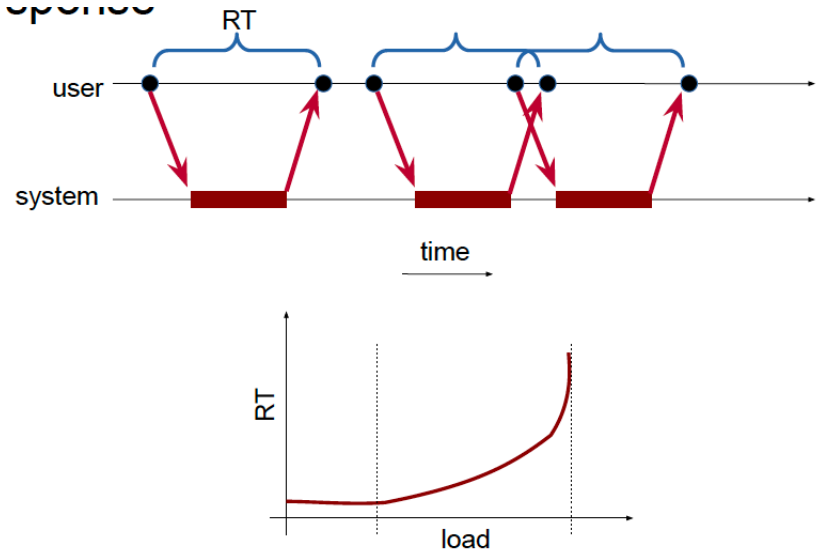
- Objetivos:
  - Métricas de performance;
  - Deteção de erros;
  - Tracking da configuração.
- Resultados:
  - Geração de alertas;
  - Apresentação gráfica.
- Exemplos: Kibana, Grafana.

8 Benchmarking

8.1 Métricas

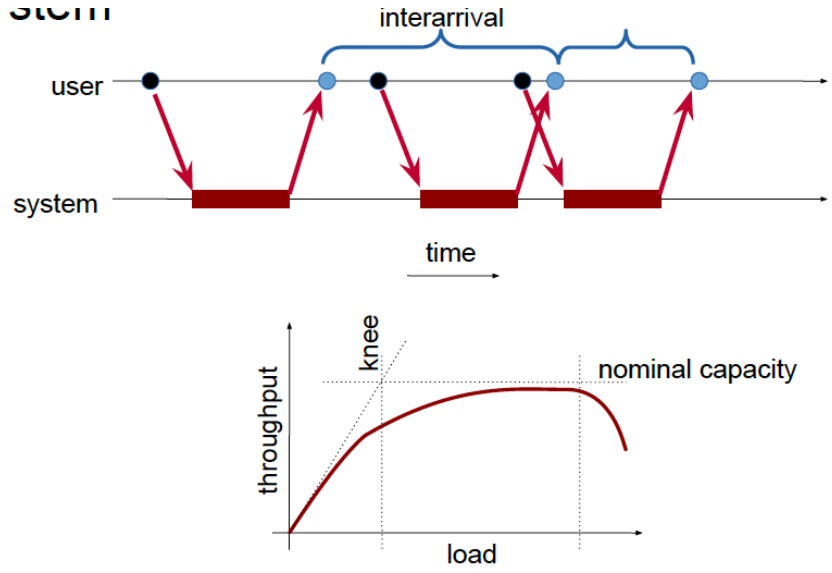
8.1.1 Response Time

Intervalo de tempo entre o pedido de um utilizador ao sistema e a sua resposta.



8.1.2 Throughput

Taxa a que as solicitações são atendidas pelo sistema.

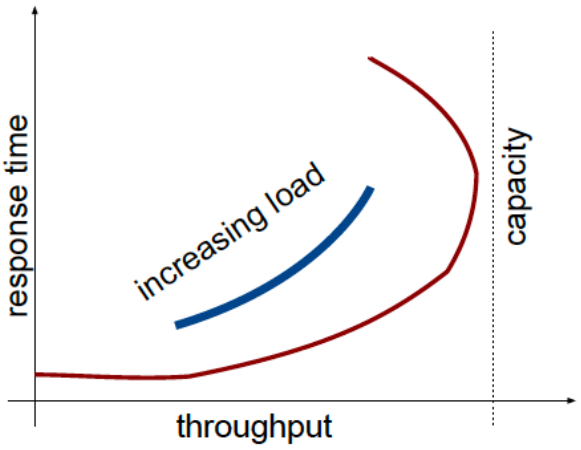


8.1.3 RT vs Throughput

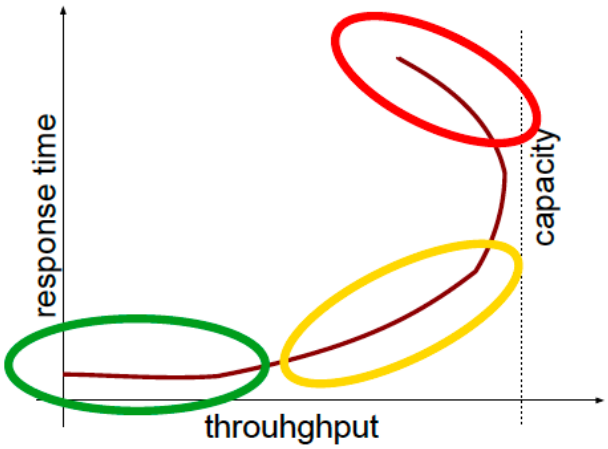
A visualização naive ($RT = \frac{1}{Throughput}$) é errada.

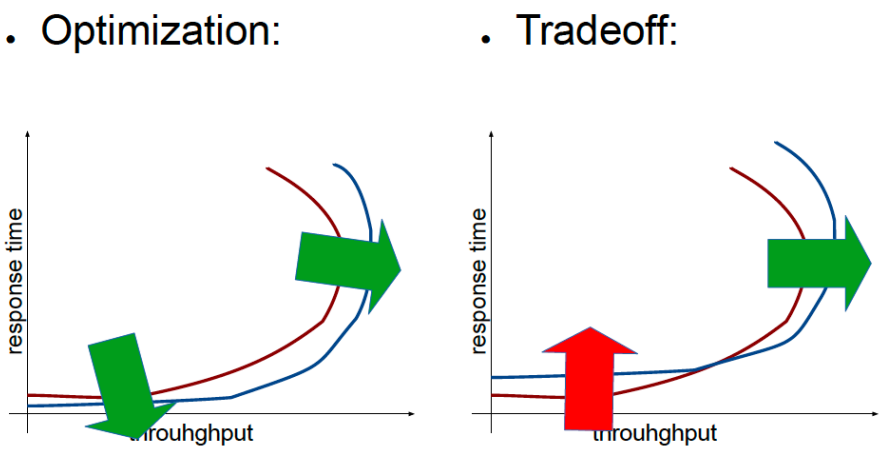
• Apenas é verdadeiro quando o sistema está ocupado 100% do tempo, com a execução de **apenas** um pedido!

A relação entre o RT e o Throughput caracteriza o desempenho do sistema:



- Inativo/Idle: As solicitações são tratadas imediatamente, pois o sistema tem capacidade extra;
- As solicitações são tratadas após uma breve espera;
- Sobrecarga: (Alguns) recursos não são usados de forma otimizada.





8.2 Outras métricas

- Utilização:
  - Recursos, por exemplo CPU, RAM, etc.
- Eficiência:
  - Relação entre rendimento e utilização.
- Confiabilidade:
  - Erros.
- Disponibilidade:
  - Tempo de ativade/inatividade.

8.3 Erros comuns

- Não definir objetivos;
- Abordagem não sistemática: difícil de reproduzir;
- workloads que não são representativas;
- Não considerar warmup nem cooldown times.

9 Teste Modelo

1. Discuta de que forma o particionamento (*sharding*) e replicação de dados podem contribuir para um melhor desempenho e tolerância a faltas de um sistema distribuído.

Sharding é um método de distribuição de um único conjunto de dados em várias bases de dados, que podem ser armazenadas em diferentes máquinas. Isto permite que conjuntos de dados maiores sejam divididos em partes menores e armazenados em vários bases de dados, aumentando a capacidade total de armazenamento do sistema.

Sharding é também uma forma de scaling conhecida como horizon scaling, uma vez que novos nós são ativados para compartilhar carga. Desta forma, ao distribuir os dados por várias bases de dados, as operações de leitura e escrita são feitas mais rapidamente, aumentando então a *performance* global do sistema. Para além disso, sharding fornece *tolerância a faltas* pois, mesmo que uma das máquinas de base de dados fique indisponível, apenas se perde uma parte dos dados, ou seja, o conjunto de bases de dados como um todo permanece parcialmente funcional.

Replicação, por outro lado, é um método de copiar a informação da base de dados para outras, isto é, consiste na criação de réplicas da base de dados primária.

Replicar dados aumenta a *performance* uma vez que é possível distribuir os pedidos pelas várias bases de dados, tendo em conta a natureza do pedido (read/write) ou até a própria geolocalização do utilizador.

Para além disso, a replicação dá uma maior tolerância a faltas, na medida em que a mesma informação está presente em vários sítios. Assim, se uma das bases de dados falhar, não se perde nenhuma informação e caso a instância primária não falhe, o sistema continua disponível.

Links:

- `https://www.mongodb.com/features/database-sharding-explained`
- `https://www.exploredatabase.com/2014/08/advantages-and-disadvantages-of-data-replication-in-distributed-databases.html`

2. Discuta quais as vantagens e desvantagens da utilização de máquinas virtuais nos modos paravirtualizado (paravirtualization) e de virtualização completa (full virtualization).

Na **Paravirtualização** o hypervisor não simula os hardware que lhe é subjacente, ou seja, o OS guest tem de ser modificado. Estas alterações no OS guest piora questões relacionadas com a manutenção e portabilidade das VMs. Por outro lado, as operações mais custosas são feitas no host nativo, e, por isso, a performance é melhor.

No modo **Full virtualization**, o hypervisor simula todos os dispositivos de hardware no ambiente virtual e, dado que esta simulação faz com que o OS guest não tenha de ser alterado, existe uma maior variedade de OS suportados e, por isso, a migração e portabilidade de VMs é mais fácil. Por outro lado, como todas as operações são feitas em duas camadas (VM + hypervisor), a performance é pior para operações de IO e ao nível de CPU.

3. A ferramenta Ansible fornece a noção de inventário(*inventory)* e módulo (module) para a criação de receitas de aprovisionamento. Descreve sucintamente o objetivo de cada um destes mecanismos.

Inventário é um ficheiro onde estão especificados todos os hosts para qual um playbook vai ser executado. Para além disto, permite fazer uma diferenciação dos hosts em grupos, de modo a poder executar uma task para um conjunto específico de hosts.

Módulos consiste num conjunto, ou unidade de código que tem como objetivo executar determinada função / ação. Pode ser executado numa *task* dentro de um playbook ou pela linha de comandos.

4. Na avaliação experimental de um sistema pode utilizar sequências de operações obtidas de um sistema real (*traces*) ou sintetizar uma sequência de operações. Discuta as vantagens/desvantagens de cada uma destas alternativas.

A utilização de traces do sistema para o avaliar tem a vantagem de nos dar a garantia que os cenários que estamos a testar são cenários que acontecem no "mundo real", visto que a sequência de operações que estamos a testar foi realizada pelo sistema, através do seu uso normal. Contudo, a utilização de traces pode não ser capaz de testar alguns cenários em que o sistema por algum motivo iria falhar ou ter uma má performance.

A utilização de uma sequência de instruções sintetizadas tem o benefício de permitir testar uma parte do sistema mais em específico e exaustivamente, uma vez que o "Tester" escolhe exatamente que cenários vão ser testados e avaliados. Porém, este método tem como desvantagem o facto de não conseguir testar o sistema de forma natural, isto é, não replicar o conjunto de instruções que seriam geradas através do funcionamento normal do sistema.

Portanto, o ideal é usar estas duas alternativas em conjunto quando se está a realizar uma avaliação experimental de um sistema, visto que complementam as fraquezas um do outro.

Outra resposta possível:

A sintetização de uma sequência de operações pretende simular o caminho esperado que um utilizador irá tomar numa aplicação. Neste sentido, é possível medir de forma proativa a disponibilidade e o desempenho de aplicações, assim como da sua infraestrutura.

Esta abordagem permite a identificação, teste e a otimização de novas funcionalidades a implementar, uma vez que nos ajuda a prever a integridade da aplicação antes e após a implementação dessas funcionalidades. Da mesma forma, ajuda a mitigar riscos mais cedo, uma vez que permite testar a aplicação antes que esta seja lançada para os utilizadores.

Por outro lado, não fornece a visibilidade em tempo real do mundo real, isto é, não pode de forma absoluta e correta, indicar a experiência de um utilizador final, uma vez que é difícil avaliar todas as variáveis atípicas ou imprevisíveis que podem afetar a experiência do cliente na vida real.

A utilização de traces do sistema fornecem informações sobre como os utilizadores reais interagem com a aplicação. Esta abordagem permite identificar problemas e alocar recursos com base numa compreensão mais clara da procura no mundo real. Dado que é informação captada com base na utilização dos utilizadores, fornece visibilidade de forma contínua, sem impor uma carga aos servidores nem à própria aplicação. É também possível monitorizar toda a experiência de um utilizador final e priorizar problemas com base no impacto neste utilizador, com o objetivo de garantir uma satisfação do cliente.

Por outro lado, uma das principais desvantagens deste mecanismo tem a ver com o facto de que, se não existem utilizadores a utilizar a aplicação, é impossível retirar dados e fazer qualquer tipo de teste.

10 Temas que possam ser importantes

- Replication;
- Sharding;
- Virtual Machine;
- Hypervisor (os 2 tipos);
- Fullvirtualization; Paravirtualization;
- Containers;
- VM vs Containers.