

Questionário de Estudos: Sistemas Distribuídos, Virtualização e APIs

Disciplina: Desenvolvimento de Sistemas Distribuídos Modernos | Público-Alvo: Ciência da Computação

Este questionário reúne **20 questões conceituais e práticas** divididas em 3 grandes blocos temáticos. Ele serve como material de fixação e preparação para exames, cobrindo os fundamentos essenciais de sistemas distribuídos, containerização e comunicação entre serviços.

Bloco 1: Definições e Fundamentos de Sistemas Distribuídos

Questão 1: Conceito e Transparência

O que define formalmente um Sistema Distribuído segundo a literatura clássica (ex: Andrew Tanenbaum) e qual é o papel do conceito de 'transparência' (de localização, replicação, falha, etc.) para os usuários e aplicações finais?

Questão 2: Escalabilidade (Vertical vs. Horizontal)

Diferencie a escalabilidade vertical (Scale-up) da escalabilidade horizontal (Scale-out). Por que a arquitetura de Sistemas Distribuídos modernos prioriza o Scale-out em detrimento do Scale-up?

Questão 3: O Teorema CAP

Enuncie o Teorema CAP (Consistência, Disponibilidade e Tolerância a Partições). Por que em redes reais de computadores a Tolerância a Partições (P) é considerada inevitável, forçando os projetistas a escolherem entre Consistência (CP) e Disponibilidade (AP)?

Questão 4: Falácias da Computação Distribuída

Peter Deutsch e outros pioneiros listaram uma série de 'Falácias da Computação Distribuída'. Escolha e explique três dessas falácias (ex: 'A rede é confiável', 'A latência é zero', 'A largura de banda é infinita') e mostre como elas afetam o design do software.

Questão 5: Falhas Parciais

O que caracteriza o fenômeno da 'falha parcial' em um sistema distribuído e por que a detecção e o tratamento dessas falhas são significativamente mais complexos do que em aplicações monolíticas executadas em um único nó?

Questão 6: Comunicação Síncrona vs. Assíncrona

Compare a comunicação síncrona (ex: HTTP/REST, gRPC) com a comunicação assíncrona baseada em eventos ou mensageria (ex: RabbitMQ, Apache Kafka). Quais são os trade-offs de acoplamento, latência e resiliência entre ambas?

Bloco 2: Virtualização — VMs, Contêineres e Aplicações

Questão 7: Virtualização Baseada em Hipervisor

Explique o funcionamento da virtualização tradicional baseada em Máquinas Virtuais (VMs). Diferencie um Hipervisor Tipo 1 (bare-metal) de um Hipervisor Tipo 2 (hosted) e analise o consumo de recursos decorrente da execução de múltiplos Sistemas Operacionais convidados (Guest OS).

Questão 8: Containerização e Isolamento no Kernel

O que são contêineres e como o motor do Docker alcança o isolamento de processos sem a necessidade de emular um hardware completo? Explique o papel dos recursos do Kernel Linux conhecidos como Namespaces e Control Groups (cgroups).

Questão 9: Comparativo VM vs. Contêiner

Analise as diferenças fundamentais entre Máquinas Virtuais e Contêineres considerando os seguintes critérios: tempo de inicialização (boot), consumo de memória/CPU, tamanho da imagem/disco e grau de isolamento/segurança.

Questão 10: Imagens Multicamadas e Copy-on-Write (CoW)

Descreva a estrutura em camadas (layers) de uma imagem Docker. Como o mecanismo de Copy-on-Write otimiza o armazenamento em disco e a velocidade de compilação (build) e distribuição das imagens?

Questão 11: Docker no Windows via WSL 2

Como o Docker Desktop possibilita a execução nativa de contêineres Linux em máquinas com Windows 10/11? Explique a integração com o Windows Subsystem for Linux 2 (WSL 2).

Questão 12: Orquestração de Contêineres

Por que a execução manual de contêineres via linha de comando torna-se inviável em cenários de microsserviços em produção? Explique a função do Docker Compose para desenvolvimento e do Kubernetes (K8s) para produção.

Questão 13: Isolamento de Redes no Docker

Como funciona a rede do tipo Bridge criada pelo Docker e de que forma ela isola os contêineres do sistema hospedeiro e de outras redes virtuais?

Bloco 3: Padrões de Codificação e Serialização para APIs

Questão 14: Formatos Textuais (XML, JSON, YAML)

Compare os formatos de representação de dados XML, JSON e YAML. Apresente os casos de uso ideais de cada um, destacando aspectos como legibilidade humana, suporte a schemas (ex: XSD, JSON Schema) e verbosidade.

Questão 15: História das RPCs e o Padrão CORBA

O que foi a especificação CORBA (Common Object Request Broker Architecture)? Quais problemas de interoperabilidade entre linguagens ela tentou resolver nos anos 1990 e por que a sua complexidade levou ao seu declínio no mercado?

Questão 16: gRPC e Protocol Buffers (Protobuf)

O que é a estrutura gRPC desenvolvida pela Google e como funciona a serialização binária através do Protocol Buffers? Quais as vantagens de desempenho do gRPC em relação a APIs RESTful com JSON?

Questão 17: Contratos Fortes e IDL

O que é uma Interface Definition Language (IDL)? Como o arquivo .proto funciona como um contrato agnóstico de linguagem e de que forma a geração automática de código (stubs) acelera o desenvolvimento de microsserviços?

Questão 18: Protocolo HTTP/2 no gRPC

Por que o gRPC depende do HTTP/2 como protocolo de transporte subjacente? Explique as vantagens técnicas trazidas pelo HTTP/2, tais como multiplexação de requisições em uma única conexão TCP, header compression (HPACK) e streaming bidirecional.

Questão 19: Serialização Textual vs. Binária

Analise os trade-offs entre utilizar um formato de serialização baseado em texto (ex: JSON) versus um formato binário denso (ex: Protobuf). Considere aspectos como facilidade de debug, velocidade de parsing, consumo de largura de banda e custo de CPU.

Questão 20: Evolução de Schemas e Compatibilidade

Como o gRPC/Protobuf garante a compatibilidade retroativa e futura (backward and forward compatibility) durante a evolução de uma API sem quebrar sistemas clientes em produção (ex: uso de números de identificação/tags nos campos)?

Gabarito Resumido (Guia de Respostas do Professor)

Item / Tópico	Resposta Esperada / Ponto-Chave
1. Definição & Transparência	Coleção de computadores independentes que se apresentam aos usuários como um sistema único e coerente. Transparência oculta a distribuição física (localização, falha, migração, escala).
2. Escalabilidade	Scale-up adiciona recursos (RAM/CPU) em 1 máquina (possui limite físico e custo exponencial). Scale-out adiciona novos nós ao cluster (ilimitado e de menor custo).
3. Teorema CAP	Impossível garantir simultaneamente Consistência (C), Disponibilidade (A) e Tolerância a Partições (P). Como redes sofrem partições, a escolha real é entre CP ou AP durante uma falha.
4. Falácias da Distribuição	Redes falham (pacotes caem), possuem latência (atraso físico), e banda limitada. O código deve ser defensivo (timeouts, retries, circuit breakers).
5. Falhas Parciais	Parte do sistema falha enquanto outra funciona. Difícil de diagnosticar pois a comunicação pode ser incerta (o nó caiu ou a mensagem atrasou?).
6. Síncrono vs. Assíncrono	Síncrono (HTTP/gRPC) exige resposta imediata (alto acoplamento temporal). Assíncrono (filas/eventos) desacopla produtor/consumidor, aumentando resiliência.
7. Virtualização VM	Usam Hypervisor para criar abstração de hardware. Cada VM roda um SO completo (Guest OS), resultando em alto consumo de RAM/Disco e overhead de processamento.
8. Contêineres & Kernel	Compartilham o Kernel do SO hospedeiro. Namespaces isolam visão de processos, rede e disco; cgroups limitam consumo de hardware (CPU/RAM).
9. VM vs. Contêiner	VMs iniciam em minutos, ocupam GBs, isolamento total de hardware. Contêineres iniciam em milissegundos, ocupam MBs, isolamento em nível de SO.
10. Multicamadas & CoW	Cada instrução do Dockerfile gera uma camada somente leitura (read-only). Contêineres compartilham camadas base e escrevem apenas em uma camada topo temporária (CoW).
11. Docker no WSL 2	Roda um Kernel Linux real altamente otimizado em uma micro-VM integrada ao Windows, permitindo performance nativa de contêineres Linux.
12. Orquestração	Gerencia implantação automática, autoscaling, autorregeneração (self-healing), balanceamento de carga e configuração de rede em múltiplos nós.
13. Rede Bridge	Cria um comutador virtual privado no hospedeiro. Contêineres na mesma ponte conversam entre si e usam NAT para acessar a internet.
14. XML vs. JSON vs. YAML	XML (muito verboso, suporte a schemas complexos XSD); JSON (leve, tipos nativos simples, padrão para APIs Web); YAML (usado para arquivos de configuração).
15. Padrão CORBA	Padrão antigo de RPC neutro para objetos distribuídos. Sofreu com extrema complexidade de especificação, peso de código e incompatibilidades entre vendedores.
16. gRPC & Protobuf	Framework RPC de alta performance que serializa dados em binário compacto (.proto), reduzindo drasticamente o payload e o tempo de parse em relação ao JSON.

Item / Tópico	Resposta Esperada / Ponto-Chave
17. IDL & Stubs	O .proto descreve estruturas de dados e métodos. O compilador (protoc) gera código tipado automaticamente para C++, Java, Go, Python, etc.
18. HTTP/2 no gRPC	Permite enviar múltiplas requisições simultâneas em uma única conexão TCP (sem head-of-line blocking), reduz latência e suporta streaming bidirecional nativo.
19. Serialização Textual vs. Binária	JSON é legível por humanos e simples de testar, mas gasta mais banda e CPU. Protobuf economiza banda e CPU, mas requer compilação e ferramentas específicas.
20. Evolução de Schemas	No Protobuf, os campos são identificados por números de tag (ex: string name = 1;). Novos campos não quebram leitores antigos que ignoram tags desconhecidas.