**1. O que são containers e por que usamos Docker para desenvolvimento?**

**Containers** são unidades padronizadas de software que empacotam código, bibliotecas e dependências, permitindo que aplicativos sejam executados consistentemente em qualquer ambiente (desenvolvimento, teste e produção), independentemente das diferenças entre os sistemas operacionais ou configurações locais.

**Docker** é uma plataforma de software amplamente utilizada para criar, gerenciar e executar containers. As razões pelas quais utilizamos Docker são:

* **Consistência:** Garante que a aplicação funcione exatamente da mesma forma em qualquer ambiente.
* **Isolamento:** Containers são isolados, impedindo conflitos entre dependências.
* **Portabilidade:** Permite executar aplicações em qualquer ambiente compatível com Docker sem ajustes adicionais.
* **Facilidade de Escala:** Containers são leves, podendo ser rapidamente iniciados, interrompidos ou replicados.
* **Velocidade e eficiência:** O uso de containers reduz significativamente o tempo para inicializar ambientes, facilitando o desenvolvimento rápido.

**2. Qual a função do PHP-FPM e por que ele é utilizado junto ao Nginx?**

**PHP-FPM (FastCGI Process Manager)** é uma implementação do protocolo FastCGI, responsável por interpretar scripts PHP de maneira eficiente e escalável.

* **Função principal do PHP-FPM:**
  + Recebe as requisições para scripts PHP, processa o código e devolve a resposta ao servidor web.
  + Gerencia pools de processos PHP, otimizando o uso de recursos (memória e CPU).
* **Razão da utilização junto ao Nginx:**
  + O Nginx não tem suporte nativo para processamento de scripts PHP diretamente (ao contrário do Apache com mod\_php). Por isso, utiliza o PHP-FPM para executar scripts PHP por meio do protocolo FastCGI.
  + Essa combinação (Nginx + PHP-FPM) é altamente eficiente, escalável e recomendada em ambientes que exigem alto desempenho, baixa latência e uso eficiente de recursos.

**3. Qual a diferença entre um servidor Apache e um servidor Nginx?**

|  |  |  |
| --- | --- | --- |
| **Característica** | **Apache** | **Nginx** |
| Arquitetura | Baseado em threads/processos (worker por requisição) | Baseado em eventos e assíncrono, não bloqueante (worker suporta múltiplas requisições simultaneamente) |
| Desempenho | Alto consumo de memória em alta carga | Menor consumo de memória, ótimo desempenho em alta carga |
| Manipulação de Requisições | Executa scripts PHP diretamente através de módulos embutidos (mod\_php) ou via FastCGI | Necessita de um interpretador externo como PHP-FPM via FastCGI |
| Configuração | Complexa e flexível com arquivos .htaccess por diretório | Centralizada, clara, porém menos granular, sem suporte nativo a .htaccess |
| Uso típico | Sites simples ou hospedagem compartilhada, flexível e fácil | Aplicações escaláveis, com alto tráfego e servidores de conteúdo estático |

Em resumo, o Nginx tende a ser mais eficiente para sites com tráfego elevado e distribuição de conteúdo estático, enquanto o Apache pode oferecer maior simplicidade e flexibilidade em ambientes menores ou hospedagem compartilhada.

**4. Como o Docker organiza imagens e containers? Explique o conceito de multi-stage builds.**

**Docker organiza imagens e containers da seguinte maneira:**

* **Imagens** são pacotes leves, imutáveis e portáveis contendo código da aplicação, bibliotecas, dependências e definições de configuração. Elas são armazenadas em camadas, permitindo reaproveitamento eficiente do espaço em disco e desempenho nas construções subsequentes.
* **Containers** são instâncias de execução de imagens Docker. Um container é criado a partir de uma imagem e representa um ambiente isolado onde a aplicação é executada.

**Multi-stage builds** é um recurso do Docker que permite criar imagens menores e otimizadas em etapas distintas:

* Permite separar as etapas de build (por exemplo, compilação de código fonte) das etapas finais de execução.
* Apenas artefatos necessários são copiados para a imagem final, reduzindo significativamente o tamanho da imagem e aumentando a segurança, removendo ferramentas ou arquivos intermediários que não são necessários na execução final.