Nome: . . . . . *Número:*. . . . .

ENGENHARIA INFORMÁTICA

Aplicações e Serviços de Computação em Nuvem

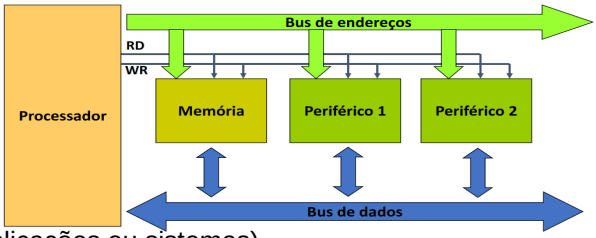
Recurso

16 de Janeiro de 2024 Duração: 1h30min

---

1 Uma arquitetura distribuída do tipo **BUS** permite uma maior flexibilidade na comunicação entre diferentes aplicações. **Descreva**, por suas palavras, em que consiste esta arquitetura e **justifique** se concorda com a afirmação anterior.

A arquitetura distribuída do tipo BUS é uma configuração em que todos os componentes ou aplicações de um sistema distribuído se comunicam através de um barramento comum, ou seja, um canal central de comunicação. Nesse modelo, os diferentes sistemas ou serviços podem se conectar ao barramento para enviar e receber mensagens. A ideia é que o barramento atue como um meio de comunicação único, o que pode reduzir a complexidade de interações entre diferentes sistemas, já que eles não precisam se conectar diretamente uns aos outros.



Como funciona:

Barramento Central: Todos os componentes (como servidores, aplicações ou sistemas) se conectam ao barramento.

Mensagens: Os dados ou mensagens são transmitidos pelo barramento, e os sistemas interessados podem "escutar" ou "ler" as mensagens que são relevantes para eles.

Desacoplamento: Cada sistema ou aplicação não precisa conhecer diretamente os outros sistemas, apenas o barramento, o que pode simplificar a arquitetura.

Flexibilidade na comunicação:
A afirmação de que a arquitetura de BUS oferece maior flexibilidade na comunicação entre diferentes aplicações é parcialmente verdadeira. Isso ocorre porque o barramento central permite que novas aplicações ou componentes sejam adicionados ao sistema de forma mais fácil, sem a necessidade de modificações complexas em cada componente. Ou seja, os sistemas podem comunicar-se entre si de forma mais genérica, por meio do barramento, sem que haja necessidade de ligações diretas.

Eu concordo com a afirmação de que a arquitetura BUS permite flexibilidade, mas essa flexibilidade vem com algumas considerações:

Escalabilidade: Em sistemas muito grandes, o barramento pode se tornar um gargalo, limitando a escalabilidade, já que todo o tráfego de dados precisa passar por ele.

Desempenho: A comunicação via barramento pode ser menos eficiente em sistemas com alta carga, uma vez que há concorrência pelo uso do barramento.

Acoplamento: Embora o barramento forneça desacoplamento, o sistema ainda depende de uma infraestrutura centralizada, o que pode ser um ponto de falha único.

Portanto, a arquitetura de BUS pode ser muito útil para sistemas pequenos ou médios onde a flexibilidade e simplicidade de comunicação são mais importantes do que a escalabilidade e o desempenho em larga escala. Em sistemas grandes e de alto desempenho, outras arquiteturas, como microservices ou arquiteturas orientadas a eventos, podem ser mais adequadas.

Nome: . . . . . .

ENGENHA

Aplicações e Serviços

16 de Janeiro de 2024

Exemplificando com o Swap:

Se estivermos instalando e configurando a aplicação **Swap** (que geralmente envolve a criação de partições ou arquivos de swap e a modificação de configurações do sistema), um role Ansible dedicado a isso pode incluir:

- **tasks**: Tarefas para criar a partição de swap ou arquivo swap, ativar o swap no sistema e garantir que ele seja persistente após reinicialização.
- **vars**: Variáveis para definir o tamanho do swap ou o tipo de arquivo (se for um arquivo de swap ou uma partição).
- **files**: Arquivos de configuração do sistema, se necessários.
- **handlers**: Reiniciar serviços relevantes (como o serviço de gerenciamento de discos ou configuração de swap) após a criação ou configuração.

2 No guião das aulas práticas dedicado à ferramenta *Ansible* foram criados diferentes **roles** para facilitar a instalação automática da aplicação *Swap*. Qual a função dos **roles** num *playbook Ansible*? De que forma os **roles** contribuem para um melhor aprovisionamento de aplicações? **Justifique** a sua resposta.

No Ansible, os roles desempenham um papel fundamental na organização e modularização dos playbooks, facilitando o aprovisionamento automatizado de aplicações e a gestão de configurações.

Os roles no Ansible são estruturas predefinidas que ajudam a organizar e reutilizar código dentro de um playbook. Eles dividem a configuração de um sistema ou a instalação de uma aplicação em várias partes lógicas e independentes, cada uma com a sua responsabilidade.

Estrutura de um Role:

tasks/: Contém as tarefas a serem executadas (por exemplo, instalação, configuração, ou execução de comandos).
handlers/: Tarefas que são acionadas por outros eventos, como reiniciar um serviço após a alteração de configuração.
defaults/: Define variáveis padrão para o role.
vars/: Contém variáveis específicas para o role que podem ser sobrescritas, se necessário.
files/: Arquivos que são copiados para o destino, como scripts ou arquivos de configuração.
templates/: Modelos de arquivos (geralmente com Jinja2) que são processados e copiados para o destino.
meta/: Contém informações sobre o role, como dependências de outros roles.

Como os Roles Contribuem para um Melhor Aprovisionamento de Aplicações:

Modularização e Reusabilidade:
Os roles permitem dividir um playbook grande e complexo em pequenas unidades funcionais. Cada role pode ser reutilizado em diferentes playbooks ou em diferentes projetos.
Facilidade de Manutenção:
Quando as configurações ou os procedimentos de instalação mudam, as alterações podem ser feitas dentro de um único role, sem impactar todo o playbook. Isso torna a manutenção do código mais fácil e menos propensa a erros.

Facilidade de Manutenção:
Quando as configurações ou os procedimentos de instalação mudam, as alterações podem ser feitas dentro de um único role, sem impactar todo o playbook. Isso torna a manutenção do código mais fácil e menos propensa a erros.

Organização e Clareza:
A utilização de roles melhora a organização do código, tornando os playbooks mais legíveis e claros. Isso é especialmente útil em ambientes complexos ou quando se trabalha em equipe.
Gerenciamento de Dependências:
Os roles podem ter dependências entre si, o que significa que se um role precisar de outro para funcionar corretamente, o Ansible pode gerenciar isso automaticamente. Isso ajuda a garantir que as aplicações sejam provisionadas corretamente, sem a necessidade de uma configuração manual.
Customização e Flexibilidade:
Os roles permitem que você defina variáveis específicas para personalizar a configuração de uma aplicação. Por exemplo, se o Swap precisar de tamanhos de partição diferentes em máquinas diferentes, isso pode ser configurado usando variáveis dentro do role.

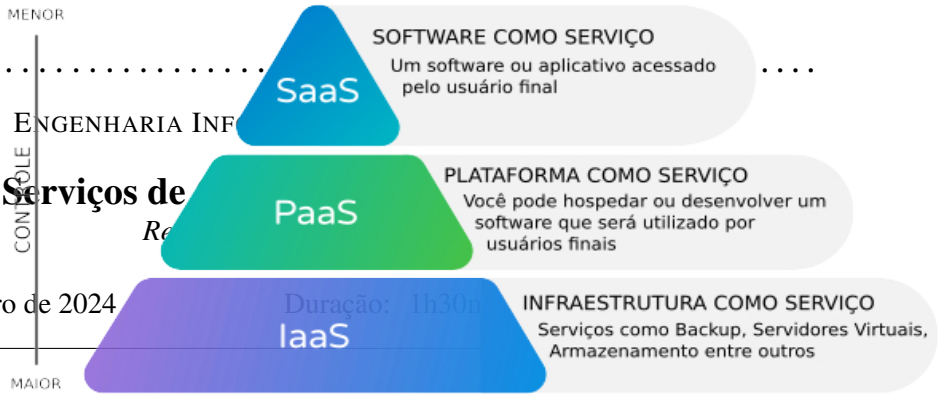
Nome: . . . . . .

ENGENHARIA INF

Aplicações e Serviços de

Re

16 de Janeiro de 2024 Duração: 1h30r



3 Diga o que entende por *elasticidade* de um serviço a correr num ambiente de computação em nuvem. Em que medida é que uma arquitetura *PaaS* contribui para a atingir? **Justifique** a sua resposta.

A elasticidade de um serviço em um ambiente de computação em nuvem refere-se à capacidade do sistema de ajustar dinamicamente os recursos computacionais (como capacidade de processamento, memória, armazenamento, etc.) em resposta às variações na demanda. Ou seja, a elasticidade permite que os recursos aumentem ou diminuam automaticamente, conforme necessário, garantindo que a aplicação ou serviço tenha a quantidade exata de recursos em tempo real, sem subutilização ou sobrecarga.

Características da Elasticidade:

Aumento Automático de Recursos: Quando a demanda por um serviço aumenta (por exemplo, mais usuários acessando uma aplicação), os recursos (como instâncias de servidores, CPU ou memória) podem ser escalados automaticamente para garantir que o serviço continue funcionando sem degradação de desempenho.

Redução Automática de Recursos: Quando a demanda diminui (por exemplo, após um pico de tráfego), o sistema pode reduzir os recursos alocados, evitando desperdício e otimizando os custos.

Escalabilidade Vertical e Horizontal:

Escalabilidade Vertical: Aumento da capacidade de uma instância (como mais CPU ou RAM).

Escalabilidade Horizontal: Adição ou remoção de instâncias de servidores (como máquinas virtuais ou containers).

Como uma Arquitetura PaaS Contribui para a Elasticidade:

PaaS (Platform as a Service) é uma camada de serviço em nuvem que fornece uma plataforma completa para desenvolver, executar e gerenciar aplicativos, sem a necessidade de gerenciar a infraestrutura subjacente (como servidores ou sistemas operacionais). As principais formas em que uma arquitetura PaaS contribui para a elasticidade incluem:

Gerenciamento Automático de Recursos:

Plataformas PaaS geralmente oferecem ferramentas integradas que permitem o ajuste automático de recursos com base na demanda do tráfego. Por exemplo, se a aplicação começa a sofrer mais requisições, o PaaS pode automaticamente provisionar novas instâncias ou aumentar a capacidade das existentes.

Esse ajuste pode ser feito sem intervenção manual, o que é um dos principais benefícios da elasticidade, pois garante que os recursos sejam sempre otimizados para o desempenho, sem desperdício.

Escalabilidade Horizontal:

Em um ambiente PaaS, é comum que a arquitetura seja orientada para a escalabilidade horizontal, ou seja, a adição ou remoção de instâncias de aplicação (como containers ou máquinas virtuais). Quando a demanda aumenta, novas instâncias podem ser automaticamente lançadas. Quando a demanda diminui, as instâncias podem ser automaticamente removidas.

Provisionamento sob Demanda:

Em um ambiente PaaS, o usuário não precisa se preocupar com o provisionamento de hardware ou a configuração de máquinas virtuais individuais. A plataforma PaaS cuida disso de forma automática, permitindo que os recursos sejam alocados e liberados com base na carga real da aplicação.

Isso significa que, durante períodos de tráfego baixo, os recursos podem ser reduzidos para minimizar custos, e durante picos de demanda, os recursos podem ser ampliados para garantir a performance da aplicação.

Capacidade de Resposta Rápida a Mudanças:

O PaaS é projetado para lidar com mudanças rápidas na demanda de recursos. Isso é especialmente útil em ambientes onde a carga pode variar drasticamente (como aplicações com tráfego sazonal ou imprevisível). O ajuste dinâmico de recursos, sem a necessidade de intervenção manual, é um dos aspectos que torna a elasticidade eficiente e prática.

Pagamentos Sob Demanda:

A elasticidade também está relacionada à cobrança sob demanda. Plataformas PaaS cobram normalmente com base nos recursos efetivamente consumidos (como tempo de execução, uso de CPU, memória, etc.), o que significa que o usuário paga apenas pelo que realmente utiliza. Isso proporciona um alto nível de eficiência de custos, já que o custo varia conforme o uso real dos recursos.

Nome: . . . . . *Número:*. . . . . .

ENGENHARIA INFORMÁTICA

Aplicações e Serviços de Computação em Nuvem

Recurso

16 de Janeiro de 2024 Duração: 1h30min

4 Para replicar o **servidor aplicacional** da aplicação *Laravel.io*, utilizada no trabalho prático, não basta apenas criar vários pods *Kubernetes* deste servidor. **Indique** se concorda com a afirmação anterior, **justificando** a sua resposta.

Concordo com a afirmação. Apenas criar vários pods Kubernetes do servidor aplicacional da aplicação Laravel.io não é suficiente para replicar completamente o servidor de forma funcional e eficaz. Justifico essa resposta com base nos seguintes pontos:

1. Necessidade de Configurações Compartilhadas:

Aplicações como o Laravel.io frequentemente dependem de recursos compartilhados, como um banco de dados, sistema de cache (Redis, Memcached) ou armazenamento de arquivos (para uploads, logs, etc.).
Apenas criar vários pods do servidor aplicacional não garante que esses serviços estejam configurados corretamente para atender à aplicação de forma distribuída.

2. Balanceamento de Carga:

Para distribuir o tráfego entre os pods de forma eficiente, é necessário configurar um serviço de balanceamento de carga no Kubernetes, como um Service com tipo LoadBalancer ou um Ingress Controller.
Sem isso, os usuários podem acessar apenas um pod ou enfrentar inconsistências no acesso à aplicação.

3. Persistência de Dados:

Laravel pode gerar arquivos temporários, logs ou depender de uploads do usuário. Esses dados precisam ser armazenados em volumes persistentes (PVs) compartilhados, já que os pods são efêmeros.
Apenas criar pods sem configurar volumes persistentes pode levar à perda de dados ou falhas de consistência.

4. Escalabilidade e Comunicação:

Criar múltiplos pods é útil para escalabilidade, mas é essencial garantir que eles possam se comunicar com outros componentes do sistema (como o banco de dados) e que as conexões sejam escaláveis para lidar com o aumento de tráfego.

Conclusão:

Replicar o servidor aplicacional Laravel.io de maneira funcional exige mais do que simplesmente criar múltiplos pods no Kubernetes. É necessário configurar serviços adicionais, como balanceamento de carga, armazenamento persistente e integração com bancos de dados e sistemas de cache, além de garantir a comunicação eficiente entre os componentes.