# Aplicações em Nuvem

Praticamente quaisquer aplicações podem ser empregadas em nuvem, havendo muito poucas restrições, como estarem limitadas às arquiteturas x86 como vimos na trilha anterior. Mas o uso de serviços em nuvem permite ainda explorar uma série de benefícios dessa arquitetura e que mudaram não só a forma das aplicações como a forma de produzi-las. Aqui vamos nos concentrar em entender essas mudanças e quais são as características do desenvolvimento e das *Aplicações Nativas em Nuvem*.

## Aplicações Tradicionais *versus* Nativas em Nuvem

Embora quase quaisquer aplicações possam ser *portadas* para nuvem, portar aplicações tradicionais não explora todos os benefícios das Soluções em Nuvem.

| Características | Aplicações Tradicionais | Aplicações Nativas em Nuvem |
|---|---|---|
| FOCO | Longevidade e estabilidade | Rápida disponibilização no mercado |
| METODOLOGIA DE DESENVOLVIMENTO | Sequenciais e em cascata | Ágil e DevOps |
| EQUIPES | Equipes isoladas de desenvolvimento, operações | Equipes colaborativas de DevOps |
| CICLOS DE ENTREGA | Longo | Curto e contínuo |
| ARQUITETURA DE APLICAÇÕES | Fortemente acoplada | Levemente acoplada |
| | Monolítica | Baseada em serviços |
| | | Comunicação baseada em APIs |
| INFRAESTRUTURA | Voltada a servidor | Voltada a containers |
| | Desenvolvida para instalação on-premise | Desenvolvida para instalação em Nuvem |
| | Dependente da infraestrutura | Portátil para qualquer infraestrutura |
| | Escalável verticalmente | Escalável horizontalmente |
| | Pré-provisionada para picos de capacidade | Capacidade sob demanda |

O quadro acima aponta as principais diferentes entre aplicações tradicionais e aplicações nativas em nuvem, bem como na forma como são desenvolvidas.

Vale a pena você entender esses pontos.

## Foco

| Características | Aplicações Tradicionais | Aplicações Nativas em Nuvem |
|---|---|---|
| FOCO | Longevidade e estabilidade | Rápida disponibilização no mercado |

O mercado sofreu grandes mudanças nos últimos anos. A transformação digital trouxe uma dinâmica muito maior nos negócios do que havia antes. Os competidores do mercado têm de ser muito mais ágeis em disponibilizar novos produtos, fazer novas ofertas, entregar novas páginas de conteúdo, disponibilizar um novo serviço etc. Essa mudança dos negócios também se refletiu na forma e no desenvolvimento das aplicações. Aplicações tradicionais visavam muito mais a estabilidade da aplicação (por exemplo, ausência de falhas) e sua longevidade visando preservar o investimento nessas aplicações. Mas *somente* a estabilidade das aplicações não atende a demanda do mercado pela rápida disponibilidade de novos serviços (e, portanto, novas aplicações!). Com isso o foco hoje das aplicações é muito mais a entrega de novos serviços (uma funcionalidade ou mesmo uma aplicação inteira) que a estabilidade.

Neste ponto os Serviços em Nuvem são uma ferramenta útil pois possibilitam a disponibilidade de infraestrutura para novos serviços de forma virtualizada, rápida e sob demanda, e você pode imaginar que o tempo a alocar recursos físicos para uma nova aplicação ou funcionalidade (servidores, espaço em disco etc.) em um modelo tradicional seria um grande limitador para essa rapidez.

Mas como fica o objetivo de estabilidade que havia nas aplicações tradicionais? A ideia é que estamos dispostos a sacrificar uma parte da estabilidade das aplicações por uma maior agilidade na entrega de novos serviços. Mudanças, como a introdução de um novo serviço, uma aplicação nova ou uma nova funcionalidade em uma aplicação já existente, são sempre potenciais fontes de instabilidade (imagine a troca da página principal de um site de e-commerce incluindo um no mecanismo de busca de produtos). Elas podem apresentar um erro não previsto, mal funcionamento, degradação de desempenho etc. Mas, do mesmo modo que novas funcionalidades podem ser feitas de forma rápida, elas podem também ser corrigidas ou retornadas rapidamente anulando, ou

minimizando significativamente, o seu impacto. Um novo e melhor mecanismo de busca de produtos em um site de e-commerce pode trazer várias vantagens para os consumidores e potencializar maiores vendas, seria aceitável então, por exemplo, uma falha de 1% nos acessos ao novo buscador, em que bastaria ao cliente refazer a busca, desde que o erro fosse rapidamente corrigido.

Essa alteração de *foco* é, de fato, o grande motor que acaba direcionando a forma e o modo de desenvolvimento das aplicações nativas em nuvem. Tanto a arquitetura das aplicações nativas em nuvem como suas metodologias de desenvolvimentos são guiadas com esse objetivo de rapidez, que só é possível graças a disponibilidade dos serviços virtualizados em nuvem.

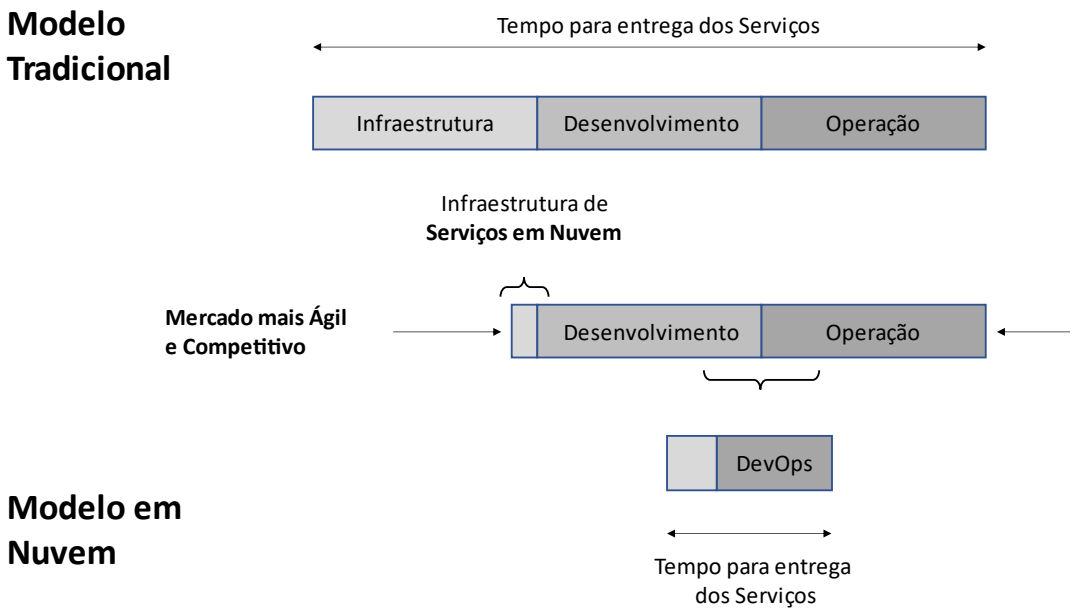
## Metodologias Ágeis de Desenvolvimento e DevOps

| Características | Aplicações Tradicionais | Aplicações Nativas em Nuvem |
|---|---|---|
| METODOLOGIA DE DESENVOLVIMENTO | Sequenciais e em cascata | Ágil e DevOps |
| EQUIPES | Equipes isoladas de desenvolvimento, operações | Equipes colaborativas de DevOps |
| CICLOS DE ENTREGA | Longo | Curto e contínuo |

Aplicações tradicionais muitas vezes seguem metodologias de desenvolvimento sequenciais, ou em cascata, com longas fases de levantamento de requisitos com o usuário, desenho lógico da aplicação, modelo de dados, codificação, testes etc. Há também uma diferenciação bastante grande entre as equipes de desenvolvimento e operações. A equipe de desenvolvimento está focada em criar uma aplicação funcionalmente correta, que atenda os requisitos do usuário. Codificada e testada, fica a cargo da equipe de operações, implantar e manter a operação diária da aplicação, disponibilizando o aplicativo para os usuários. Nesse modelo era bastante comum a disponibilidade final de novas aplicações após meses de desenvolvimento.

Aplicações nativas em nuvem requerem abordagens de entregas contínuas, desenvolvimento ágil e integração dos times de desenvolvimento e operação para reduzir o tempo de implantação dos serviços em produção.

Esse longo prazo para a disponibilidade final de novas aplicações era talvez aceitável em um mercado menos competitivo e menos digital, e colaborava para isso o tempo para disponibilidade de infraestrutura (como servidores, softwares, armazenamento etc.) que, sem formas de virtualização ou de serviços em nuvem, também precisavam aguardar meses para serem instalados, configurados etc. Um mercado mais ágil e competitivo, impulsiona então mudanças nessa forma de desenvolvimento, e na ausência de longos prazos para a disponibilidade de infraestrutura com o uso de serviços em nuvem, coloca todo o prazo de entrega de aplicações nos ciclos de desenvolvimento e operação, que precisam assim tornarem-se mais ágeis.




Explorar as diferentes metodologias de desenvolvimento de software e de integração (DevOps) é muito mais amplo do que cabe em uma Introdução aos Serviços em Nuvem. De qualquer modo podemos dar uma rápida definição desses conceitos para atender nossos propósitos aqui e para que você possa explorá-los melhor depois.

### ➔ O que é Desenvolvimento Ágil de Software?

Desenvolvimento ágil de software é um termo abrangente para um conjunto de estruturas e práticas baseadas nos valores e princípios expressos no Manifesto para Desenvolvimento Ágil de Software. A metodologias de desenvolvimento ágil de software em geral envolvem práticas comuns como programação em pares, desenvolvimento orientado a testes, *stand-up meetings*, sessões de planejamento e *sprints,* com participação do usuário, entregas parciais e contínuas com **MVP (*mínimo valor do produto*)** etc. e algumas das metodologias comumente empregadas são **Scrum** , **Extreme Programming** e **Feature-Driven Development (FDD)**.

Em um mercado cada vez mais dinâmico, essas metodologias nascem como uma resposta para a constante mudança de requisitos e proporcional entregas de valor constantes e rápidas.

Tradicionalmente equipes de Desenvolvimento e Operações são áreas diferentes e apresentam motivações e objetivos diferentes. O Desenvolvedor quer entregar novas funcionalidades e serviços, corrigir bugs e fazer mudanças (todos potenciais de instabilidade) e a Operação quer os serviços de produção estáveis e funcionando corretamente. Do mesmo modo que os times de desenvolvimento evoluem com a adoção de metodologias ágeis para estarem alinhados às novas necessidades do negócio, a operação também é forçada a adaptar os seus métodos. Para fazer frente a maior velocidade de desenvolvimento das aplicações, a operação e os serviços de infraestrutura precisam se integrar ao desenvolvimento fazendo uso das mesmas práticas de desenvolvimento de software, tratando **infraestrutura como código** (o que é possível graças a virtualização). Desse modo, configurações de software como um banco de dados ou um servidor Web, passam a ser tratadas do mesmo modo que o código de uma aplicação: são versionados, sujeitos a *roll back* e distribuídos como código.

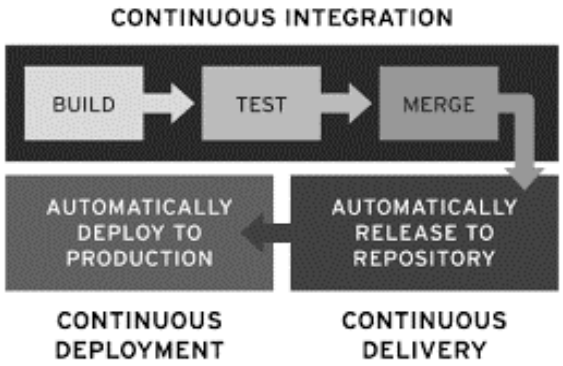
## O que é DevOps?

Devops é um termo que envolve um **conjunto de práticas de integração de equipes** de desenvolvimento, de operações (infraestrutura e administradores de sistemas) e de apoio (controle, segurança), todas envolvidas na implantação de Aplicações, e a **adoção de processos automatizados para produção rápida e segura de aplicações e serviços**.

A cultura DevOps sustenta-se em três pilares principais:

- **CI Integração Contínua**: Transferência de conhecimento e experiências entre as áreas de Desenvolvimento, Operações e Apoio.
- **CD Deploy e Delivery (Implantação) Contínuos**: liberação rápida e continua de novas versões de software ou serviços.
- **Feedback contínuo:** feedbacks frequentes das equipes envolvidas em todas as fases do ciclo de vida do software ou serviço.

Esses objetivos, ou pilares, somente podem ser alcançados com uma série de ferramentas que permitem tornar todas as fases do desenvolvimento e de implantação de software, desde o *build* (compilação dos pacotes), testes, etc. até o *deploy* final em produção, automatizadas e, portanto, mais padronizadas, livres de erro e mais ágeis. Essas *esteiras automatizadas* de entrega de aplicações são comumente denominadas *pipelines.*



Há vários softwares, em sua maioria Open Source, empregados na construção desses pipelines de DevOps e se você estiver interessado pode buscar informações na internet sobre produtos como **Ansible** (ferramenta de código aberto para automatizar o provisionamento de software, gerenciamento de configuração e implantação de aplicativos), **Git** (ferramenta de versionamento de código e arquivos que suporta desenvolvimento e trabalho concorrente dos membros da uma equipe), **Chef** (ferramenta de gerenciamento de configuração de código aberto), **Jenkins** (ferramenta de código aberto para automatizar o pipeline de entrega que permite testar e relatar alterações e desvios de software), **Selenium** (para automatizar testes para aplicativos da web).

Essas definições são suficientes para você ter uma ideia das diferenças em como são entregues (do seu desenvolvimento até a entrega ao consumidor final) aplicações nativas em nuvem e aplicações tradicionais.

Discutimos até agora como aplicações nativas em nuvem são criadas e como esse processo se diferencia da criação de aplicações tradicionais. Vamos daqui para diante nos concentrar no que a arquitetura e infraestrutura dessas aplicações se diferenciam.

## Arquitetura de Aplicações Nativas em Nuvem

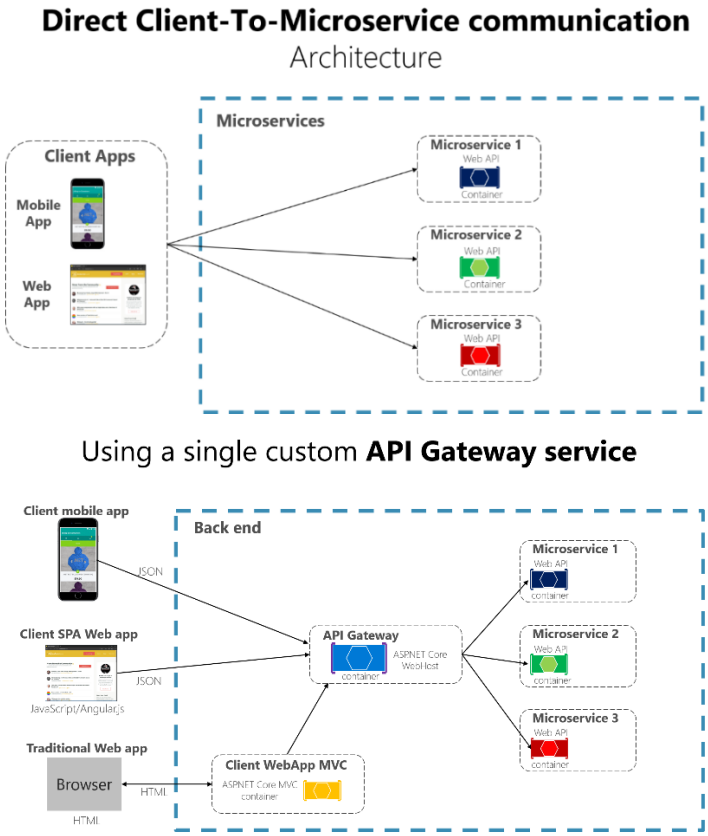
| Características | Aplicações Tradicionais | Aplicações Nativas em Nuvem |
|---|---|---|
| ARQUITETURA DE APLICAÇÕES | Fortemente acoplada | Levemente acoplada |
| | Monolítica | Baseada em serviços |
| | | Comunicação baseada em APIs |

Aplicações tradicionais são em geral construídas de modo monolítico, isto é, elas são constituídas de grandes blocos de código e que agregam grande número de funções. Isso torna as aplicações difíceis de serem mantidas e alteradas. Imagine que você precisa fazer uma alteração, em uma única funcionalidade, para corrigir um erro ou entregar uma melhoria para o usuário. Todo código é então compilado e distribuído e, outras funcionalidades, sem qualquer relação

com a sua mudança, podem ser impactadas. Isso é o que chamamos de ***acoplamento forte* da aplicação**. As funcionalidades estão acopladas e, mudanças em uma funcionalidade, não podem ser feitas sem impacto em outras. Você deve entender que recompilar um código, importar novas bibliotecas, fazer o *deploy* e instalar artefatos de software já produz mudanças na aplicação, mesmo que não tenham sido feitas alterações de código, e essas mudanças podem ter impactos que não foram previstos.

Em uma arquitetura de **microserviços** as aplicações são desmembradas em componentes mínimos e independentes. Cada microsserviço é um componente separado e são ideais para a criação de serviços mais modulares e **menos acoplados**. Uma forma simples de você pensar um microserviço é imaginar um '*programa*' (componente) que executa uma única função simples, por exemplo, um programa que retorna a lista de produtos adquiridos no último pedido de um cliente ou o preço do frete de um produto. Esses serviços podem então serem chamados por uma interface do usuário (cliente), podem se comunicar entre si, e normalmente de maneira ***stateless***: esses módulos da aplicação recebem requisições, processam e devolvem ao seu requerente o resultado, geralmente via **HTTP** (como uma chamada de internet, para cada requisição há um retorno, mas não persiste do lado do servidor o 'estado' do cliente). Nessa solução, uma modificação no serviço de lista de produtos não interfere em nada na funcionalidade de preço do frete (ou vice-versa), pois não há dependência, ou acoplamento entre elas, e podem até mesmo serem com linguagens diferentes.

Essa abordagem de microserviços pode ser feita por uma **comunicação direta da aplicação** cliente com os microsserviços, e essa abordagem é bastante empregada para pequenos serviços. Mas uma abordagem de **API Gateway** é ainda mais empregada para aplicações mais modernas e complexas, como aplicações onde o cliente é por exemplo um App (celular). Um API Gateway é um software que gerencia uma série de complexidades que ocorrem na comunicação entre o cliente e os microserviços como **segurança, persistência de sessão, agregação de serviços para redução do número de chamadas** etc. e ainda reduz o acoplamento entre o cliente e o microserviço.




E ainda múltiplos API Gateways podem ser empregados.

Em resumo, a arquitetura de aplicações nativas em nuvem é muito mais granular que as aplicações tradicionais, havendo centenas ou milhares de pequenos componentes (serviços) que se comunicam através da rede (HTTP) e executam de modo independente e de modo *stateless.* Essas aplicações são mais adequadas para sofrerem modificações contínuas, sendo menor o esforço de criação, teste e implantação dos microserviços e seu impacto nos demais componentes.

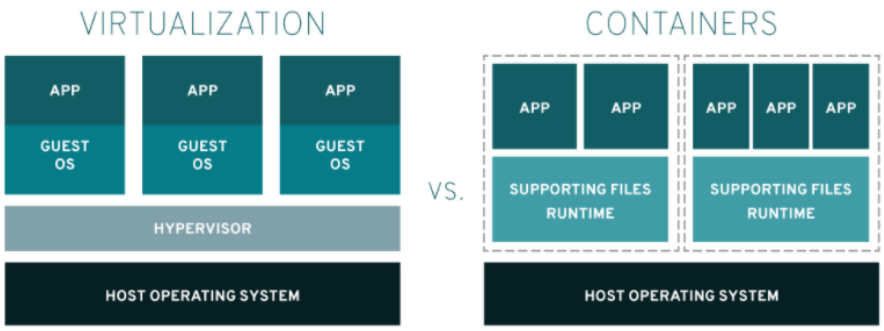
# Infraestrutura

| Características | Aplicações Tradicionais | Aplicações Nativas em Nuvem |
|---|---|---|
| INFRAESTRUTURA | Voltada a servidor | Voltada a containers |
| | Desenvolvida para instalação on-premise | Desenvolvida para instalação em Nuvem |
| | Dependente da infraestrutura | Portátil para qualquer infraestrutura |
| | Escalável verticalmente | Escalável horizontalmente |
| | Pré-provisionada para picos de capacidade | Capacidade sob demanda |

Como você viu as aplicações em nuvem são bastante diferentes das aplicações tradicionais e, também são desenvolvidas de forma diferente. O que muda agora para infraestrutura dessas aplicações?

Os microserviços são componentes relativamente bastante menores que aplicações monolíticas, são *stateless* e podem, portanto, executar empregando muito poucos recursos de processamento (processador, memória).

Um **container** é uma metodologia utilizada para empacotar aplicações para que elas possam ser executadas como todas as suas dependências (entenda como os imports de bibliotecas e serviços do sistema operacional necessários para o seu funcionamento) de maneira isolada e eficiente. Um container permite executar um microserviço (aplicação ou programa) de forma isolada e ainda permite portar esse componente, que poderá executar em qualquer máquina virtual (ou física). O container é mais um nível de virtualização em que agora não necessitamos mais de uma máquina virtual para processar uma aplicação.



Os containers armazenam então um microsserviço ou aplicação, e todos os recursos que são necessários para executá-los (as dependências). Tudo que eles contêm é mantido em arquivo chamado de **imagem** que contém todas as bibliotecas e dependências. Assim esse arquivo pode ser portado para qualquer servidor e ser executado. Como os containers são em geral pequenos (alguns MB), geralmente há centenas ou mesmo milhares desses componentes levemente acoplados executando em diferentes máquinas provendo, por exemplo, uma grande e complexa aplicação Web ou App, com o portal de vendas da Amazon, ou a NetFlix.

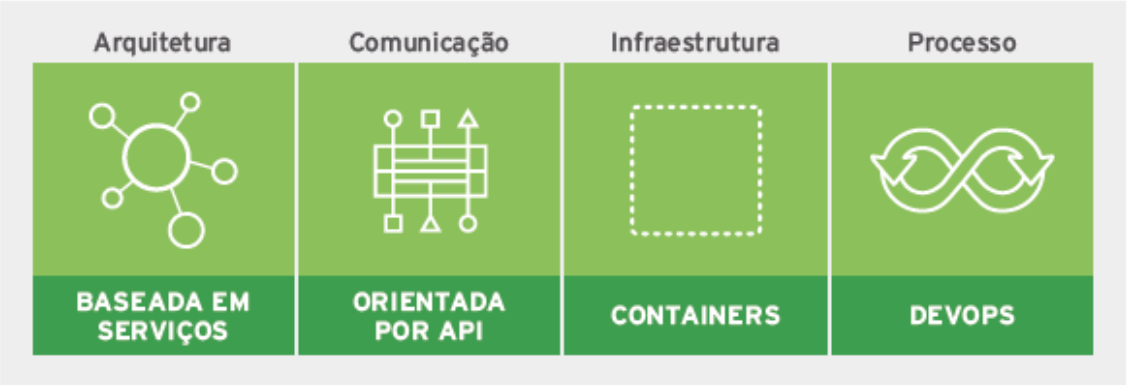
Sendo em grande número esses componentes em geral requerem um **orquestrador** para o seu bom funcionamento. Um orquestrador (para o projeto Docker o **Kubernetes** é hoje o mais empregado) é um software que gerencia esses múltiplos containers. Ele pode instanciar (criar) novos containers para atender uma maior demanda da aplicação, 'destruir' (parar a execução e eliminar) containers que apresentem erro de execução ou por que não são mais necessários, atualizar containers com uma nova versão do aplicativo ou microserviço etc.

Os containers são ideais para o desenvolvimento de aplicações modernas e nativas em nuvem. Eles suportam tanto a arquitetura das aplicações (suporte a execução de vários microserviços) como atendem também o desenvolvimento das aplicações, permitindo o deploy contínuo (com a orquestração da atualização dos containers), criação de ambientes portáveis e isolados de desenvolvimento e execução das aplicações etc. Eles ainda permitem uma grande escalabilidade horizontal das aplicações (os containers podem ser distribuídos para diversos servidores virtuais) e aumentam a eficiência do uso de recursos (com o aumento densidade de uso dos recursos das máquinas).

Você pode encontrar um nível ainda mais granular de execução de microserviços nos provedores em nuvem. O **AWS Lambda**, por exemplo, permite criar 'funções' que podem ser chamadas por um cliente através de um API Gateway sem a necessidade de definição pelo usuário de qualquer ambiente ou recursos de execução, nem mesmo um container. A execução dessa função (um programa em Python ou Java, ou Go na plataforma Google) é o 'serviço' dado pelo provedor, e é transparente para o usuário o uso de containers ou máquinas

virtuais nessa execução. Essa modalidade, entretanto, é ainda bastante limitada sobretudo para aplicações com grande número de requisições.

A figura abaixo resumo para você as principais características das aplicações nativas em nuvem que vimos.




# Sumário

Aqui você pôde ver aqui as aplicações nativas em nuvem se diferenciam bastante de aplicações tradicionais, seja em sua arquitetura ou no modo como são desenvolvidas. Aplicações nativas em nuvem são desenvolvidas para terem atualizações contínuas com um impacto mínimo dos serviços. Elas são construídas com diversos componentes de baixo acoplamento, pequenos, portáveis e que podem escalar horizontalmente para atender aumentos (ou reduções) de demandas da aplicação e aumentam a eficiência dos recursos de processamento. E é bastante coisa para aprender!

## Referências

__ **Red Hat. O caminho para a adoção de aplicações nativas em cloud.** Disponível em: https://www.redhat.com/pt-br/engage/cloud-native-application-development-20180622 Acesso: 08.02.2021.

__ **Red Hat. CONTAINERS: O que é Docker?** Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-docker Acesso: 08.02.2021.

__ **Microsoft. Padrão de gateway de API versus comunicação direta de cliente com microsserviço**. Disponível em:

https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern Acesso: 15.02.2021.

__ **Red Hat. DEVOPS: O que é CI/CD?** Disponível em:

https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd Acesso: 15.02.2021.