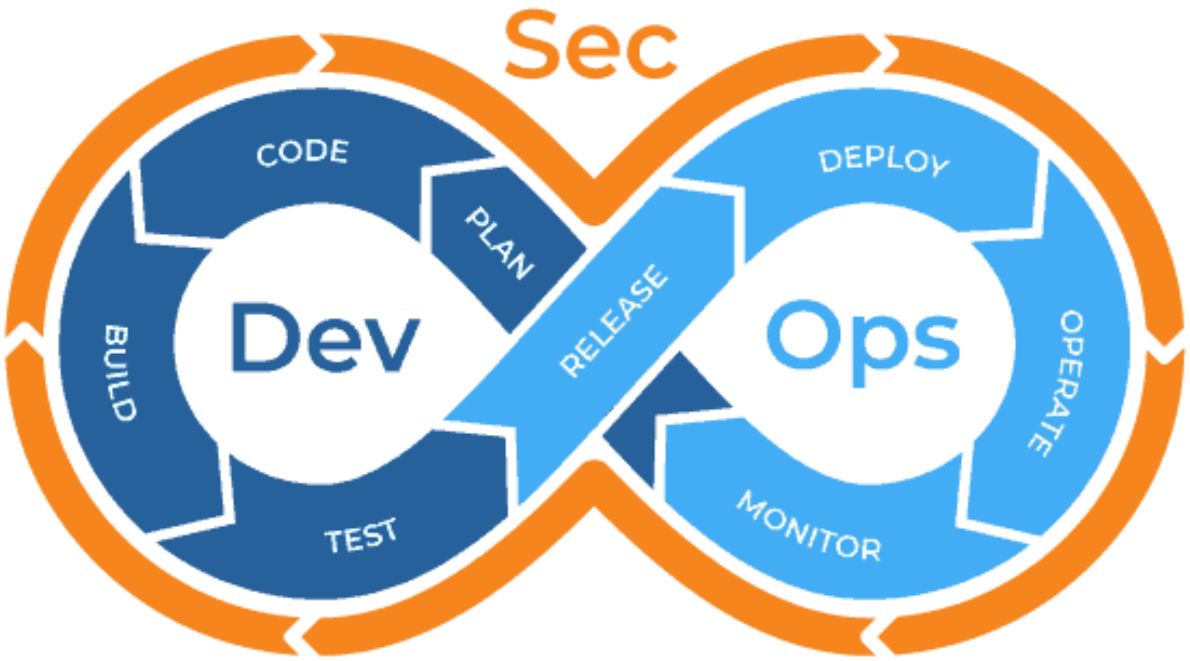
DevSecOps



Diogo Antonio Sperandio Xavier

SUMÁRIO

Introdução

A palavra **DevOps** vem de Desenvolvimento(Dev) e Operações(Ops). **DevOps é uma união entre pessoas, tecnologias e processos para continuamente agregar valor aos clientes**. Para as equipes, o DevOps proporciona uma atuação de maneira coordenada e colaborativa para gerar produtos melhores e mais confiáveis do que se trabalhassem isoladamente. Com as ferramentas e a cultura DevOps adotadas, as equipes melhoram a resposta à necessidade dos clientes, aumentam a confiança nos resultados que constroem e cumprem de maneira mais efetiva.

O objetivo do material em questão é apresentar os conceitos de DevOps a fim de facilitar o entendimento de quem está começando a programar. **Não são necessários conhecimentos em programação** para compreender os conceitos e a ideia, mas caso possua algum, fica mais fácil o aprofundamento no tema.

Espero que esse livro possa colaborar com seu desenvolvimento de alguma forma, seja com conhecimento ou com um passo-a-passo que melhore sua venda e seus negócios. Caso isso aconteça, já estou de antemão muito feliz por isso.

O que é SDLC?

Ciclo de Vida de Desenvolvimento de Software (SDLC) é um processo a ser seguido durante o projeto de um software. Consiste em um plano detalhado que descreve como projetar, desenvolver, testar e manter softwares de alta qualidade. O SDLC visa produzir um software de alta qualidade que atenda ou supere as expectativas do cliente, alcance a conclusão dentro dos prazos e estimativas de custo. SDLC fornece uma estrutura que define as tarefas executadas em cada etapa do processo de desenvolvimento de software.

Em resumo, o SDLC define uma metodologia para melhorar a qualidade do software e o processo geral de desenvolvimento.

A figura a seguir é uma representação gráfica dos vários estágios de um SDLC:

Fonte: https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc

Um ciclo de vida de desenvolvimento de software típico consiste nos seguintes estágios:

Etapa 1: Planejamento (Planning)

A análise de requisitos é a etapa mais importante e fundamental no SDLC. É realizado pelos membros seniores da equipe com insumos do cliente, do departamento de vendas, pesquisas de mercado e especialistas do setor.

Essas informações são então utilizadas para planejar a abordagem do projeto básico e realizar o estudo de viabilidade do produto nas áreas econômica, operacional e técnica.

O planejamento dos requisitos de garantia da qualidade e a identificação dos riscos associados ao projeto também são feitos na fase de planejamento. O resultado desse estudo de viabilidade técnica é definir as várias abordagens técnicas que podem ser seguidas para implementar o projeto com sucesso com riscos mínimos.

Estágio 2: Definição de Requisitos (Analysis)

Uma vez que a análise de requisitos é feita, o próximo passo é definir e documentar claramente os requisitos do produto e obtê-los aprovados pelo cliente ou pelos analistas de mercado. Isso é feito por meio de um documento que consiste em todos os requisitos do produto a serem projetados e desenvolvidos durante o ciclo de vida do projeto.

Estágio 3: Projeto (Design)

O documento de requisitos gerado no estágio anterior é a referência para que arquitetos de produtos apresentem a melhor arquitetura para o produto a ser desenvolvido. Com base nos requisitos especificados nesse documento, geralmente mais de uma abordagem de projeto para a arquitetura do produto é proposta e documentada em um novo documento.

Este novo documento é revisado por todas as partes interessadas e com base em vários parâmetros como avaliação de risco, robustez do produto, modularidade do projeto, orçamento e restrições de tempo, a melhor abordagem de projeto é selecionada para o produto.

Uma abordagem de design define claramente todos os módulos arquitetônicos do produto, juntamente com sua comunicação e representação de fluxo de dados com os módulos externos e de terceiros (se houver). O design interno de todos os módulos da arquitetura proposta deve ser claramente definido com os mínimos detalhes nesse documento gerado pela arquitetura.

Estágio 4: Desenvolvimento do Produto (Implementation)

Nesta fase do SDLC o desenvolvimento real começa e o produto é construído. O código de programação é gerado conforme definido no documento de arquitetura durante a etapa anterior. Se o projeto for realizado de forma detalhada e organizada, a geração de código pode ser realizada sem muita complicação.

Os desenvolvedores devem seguir as diretrizes de codificação definidas por sua organização e ferramentas de programação como compiladores, interpretadores, depuradores, etc. são usadas para gerar o código. A linguagem de programação é escolhida em relação ao tipo de software que está sendo desenvolvido.

Etapa 5: Teste e Integração (Testing and Integration)

Este estágio geralmente é um subconjunto de todos os estágios, pois nos modelos modernos de SDLC, as atividades de teste estão principalmente envolvidas em todos os estágios do SDLC. No entanto, esta etapa refere-se apenas à etapa de teste do produto onde os defeitos do produto são relatados, rastreados, corrigidos e retestados, até que o produto atinja os padrões de qualidade definidos no documento de requisitos.

Etapa 6: Manutenção - Implantação no mercado (Maintenance)

Uma vez que o produto esteja testado e pronto para ser implantado, ele é lançado formalmente no mercado apropriado. Às vezes, a implantação do produto acontece em etapas de acordo com a estratégia de negócios dessa organização. O produto pode ser lançado primeiro em um segmento limitado e testado no ambiente real de negócios.

Em seguida, com base no feedback, o produto pode ser lançado como está ou com melhorias sugeridas no segmento de mercado-alvo. Após o lançamento do produto no mercado, é feita sua manutenção para a base de clientes existente.

O que é a Metodologia Ágil

Metodologia ágil é uma forma de conduzir projetos que busca dar maior rapidez aos processos e à conclusão de tarefas. Não apenas isso, a metodologia ágil baseia-se em um fluxo de trabalho mais ágil, flexível, sem tantos obstáculos e com total iteratividade.

O Manifesto Ágil é composto por 4 pilares e 12 princípios.

Os 4 Pilares

Fonte: https://aelaschool.com/estrategia/tudo-sobre-filosofia-agile

1. **As interações dos indivíduos são mais importantes do que processos ou ferramentas**: devemos **priorizar a interação** entre as pessoas e questionar qualquer processo ou ferramenta que atrapalhe tais interações. Se é mais fácil sentar ao lado de uma pessoa e facilitar a conversa, por exemplo, por que o deveríamos fazê-lo por e-mail?
2. **Produtos e serviços funcionais são mais importantes do que documentação pesada**: não há sentido em perder tempo elaborando uma documentação/**especificação de um produto que não funciona direito**. É preferível ter um produto excelente e **documentar o mínimo necessário,** sem informações desnecessárias;
3. **A colaboração com o cliente é mais importante do que a negociação de contrato**: para evitar grandes frustrações, o **cliente deve colaborar com o desenvolvimento de seu produto.** É importante haver reuniões de alinhamento durante o desenvolvimento para que eventuais correções de rota sejam feitas. A **co-criação** é a palavra-chave desse pilar.
4. **Ser capaz de responder às mudanças é mais importante do que seguir um plano**: é normal surgirem **dúvidas e situações inesperadas** durante um projeto. O quarto pilar do Manifesto

coloca a importância de conseguir corrigir seu plano de projeto levando em consideração essas mudanças. Dessa forma, garantir que o produto final estará cumprindo com as expectativas do seu usuário.

Os 12 Princípios

Fonte: https://aelaschool.com/estrategia/tudo-sobre-filosofia-agile

Cada um dos princípios está relacionado com um dos pilares descritos acima.

Princípios do pilar nº 1

1. **Desenvolver os produtos e serviços envolvendo times motivados**: deixar claro os objetivos do projeto e **confiar no trabalho** da equipe;
2. **Conversas face a face**: mais interação entre as pessoas evita ruídos na comunicação, **evitando erros** no projeto;
3. **Melhores soluções emergem de times auto-organizáveis**: dar **autonomia** para os times tomarem as melhores decisões possíveis, e trabalhar com uma equipe **multidisciplinar**;
4. **Reunir para refletir e corrigir**: os times capazes de **parar para conversar** sobre o andamento do projeto são mais eficazes em efetuar as correções necessárias.

Princípios do pilar nº 2

1. **Realizar entregas relevantes em ciclos curtos**: uma maneira ágil de verificar se o desenvolvimento do projeto está cumprindo as **expectativas** é quebrar o resultado final em pequenas entregas. Para cada uma dessas entregas, o prazo deve ser o menor possível sem afetar a qualidade do produto e a qualidade de trabalho da equipe;

2. **Um produto funcional é a melhor maneira de medir o sucesso do seu time**: avaliar a equipe e remunerá-la de acordo com suas capacidades e **excelência** na entrega de produtos funcionais;
3. **Criar um ambiente onde a entrega de valor seja constante**: interação entre todas as partes do projeto – cliente, programadores e usuários – deve ser **constante**. Dessa forma, as melhorias do produto também serão constantes;
4. **Dar atenção constante aos detalhes técnicos e de Design**: a preocupação com a **qualidade das entregas** melhora a qualidade de execução dos times.

Princípios do pilar nº 3

1. **A prioridade do time é satisfazer o cliente com entregas antecipadas e contínuas, com valor percebido**: quanto mais cedo entregarmos uma parte do produto para o cliente, mais cedo ele nos dará o **feedback**;
2. **Todas as pessoas envolvidas no projeto devem trabalhar juntas**: a interação entre cliente, UX, programadores e usuários deve ser constante. Dessa forma, **elimina-se os ruídos na comunicação.**

Princípios do pilar nº 4

1. **Solicitações de mudança, mesmo que tardias, são sempre bem vindas**: não importa o momento da solicitação da mudança. O importante é garantir a **satisfação do cliente** e re-priorizar as atividades conforme forem necessárias;
2. **Simplicidade é a arte de minimizar o trabalho futuro**: planejamento e organização das prioridades. Dessa forma, **simplificamos** o método de trabalho.

Melhores práticas

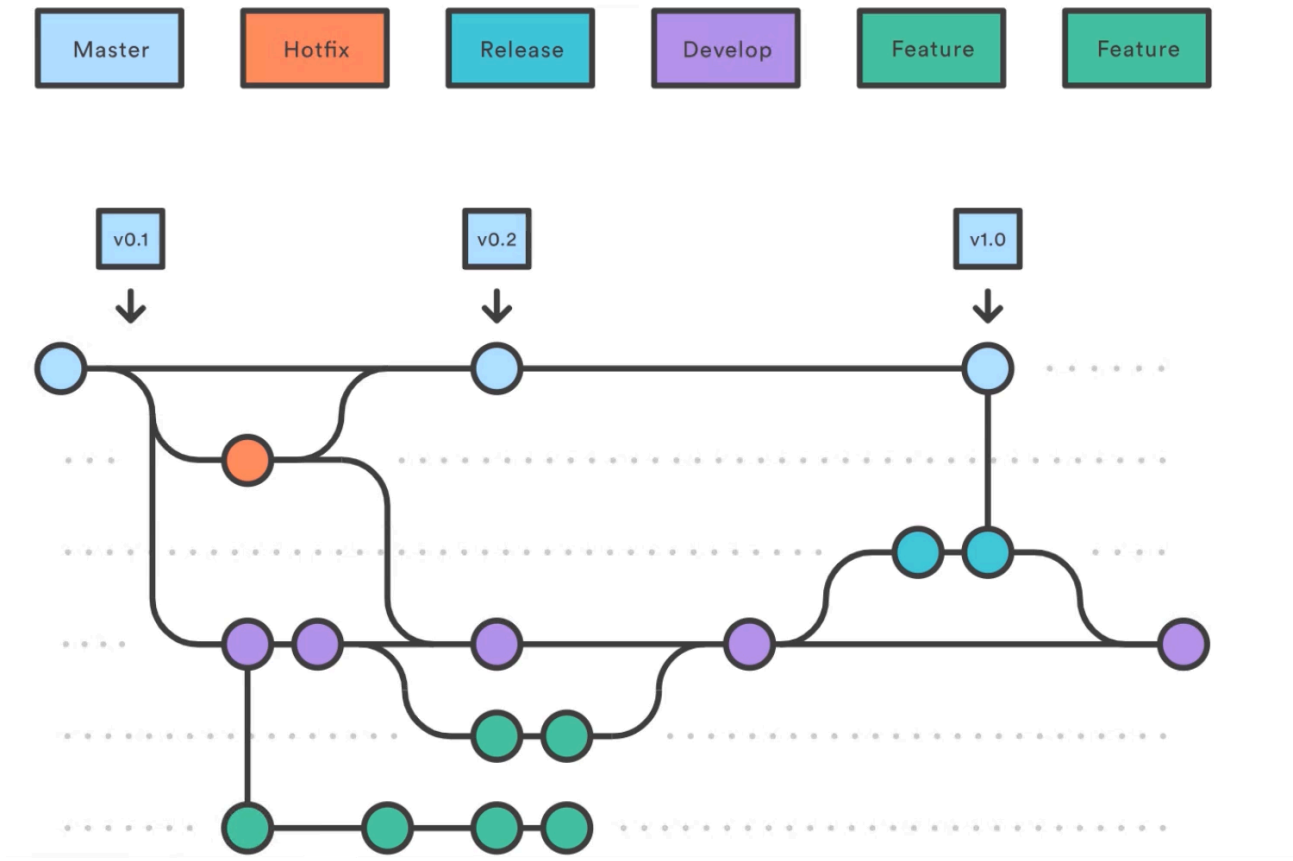
Observamos que o Manifesto Ágil reúne pilares e princípios. Mas **como aplicamos isso na rotina de trabalho?**

Confira abaixo algumas **práticas** que estão alinhadas com a filosofia Ágil.

- **Contrato Social**: é a descrição das **regras de trabalho** da sua equipe. O próprio time é quem escreve esse documento. Contempla diversas informações como formas de trabalho, horários, pausas e etc;
- **Reuniões Standup**: reuniões curtas e feitas em pé. A ideia é ser **rápido e produtivo**, sem dar margem para distrações ou interrupções;
- **Cronograma das entregas**: possibilitando quebrar o projeto em pequenos **ciclos** e entregas;

- **Retrospectiva**: discussão entre ciclos, abordando dúvidas, acertos, erros e novas ideias para o próximo ciclo de entrega;
- **Showcase**: apresentação do projeto para o cliente. Lembrando que a apresentações devem ser feitas durante todo o projeto para garantir as **expectativas** e reduzir as **frustrações**;
- **Jornada do usuário**: para entender as **interações do usuário** com o produto e **engajar a equipe** com sua história..

Gitflow



O que é CI/CD?

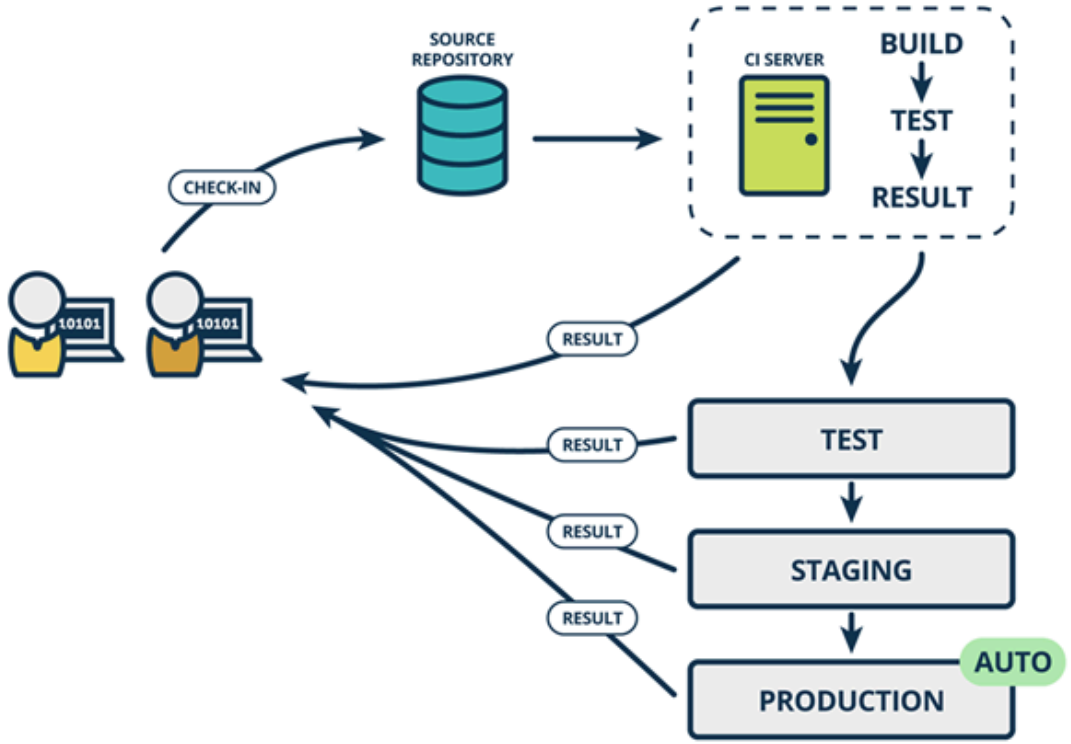
CI/CD - Continuous Integration/Continuous Delivery, ou Integração Contínua/Entrega Contínua resumidamente, representam um conjunto de práticas e ferramentas que nos ajudam a entregar software de qualidade com frequência em pequenos pedaços.

Esses dois princípios vêm do manifesto ágil, ou seja, queremos entregar software com frequência, fazendo pequenos incrementos, com novas funcionalidades e ir ampliando esse software. E a cada nova funcionalidade, quando subimos o código para o servidor de versionamento, os testes rodam, ou seja, a parte CI, e o CD, nos ajuda a colocar esse nosso software de forma automatizada nos diferentes ambientes da empresa.

A parte **CI (Continuous Integration**, integração contínua) descreve a forma como as equipes de desenvolvimento implementam e testam regularmente pequenas mudanças incrementais de código que, em seguida, são mescladas em um repositório compartilhado de controle de versões. Então, esses "checkins" são verificados por uma criação automatizada para que qualquer problema com o código possa ser identificado e resolvido rapidamente. A

CI permite que as equipes de desenvolvimento trabalhem simultaneamente no mesmo aplicativo sem criar conflitos.

A parte **CD (*Continuous Deployment/Delivery,* *implantação/entrega contínua*)** é a continuação do CI, ela possibilita a entrega rápida, sustentável e confiável do software onde ele precisa estar (independentemente de isso significar o envio dele para ambiente produção, desenvolvimento ou teste, ou diretamente para os usuários). Em outras palavras, as mudanças feitas em seu código são implantadas em um ambiente específico.



The CICD proccess - Fonte: https://www.mindtheproduct.com

Ferramentas de CI/CD

As ferramentas CI/CD estão disponíveis em variantes comerciais e de código aberto, dependendo de suas especificações e requisitos orçamentários.

Abaixo seguem algumas das principais ferramentas sendo utilizadas em 2022:

1. **Azure Pipelines:**

Visão geral: O Azure Pipelines (parte do Microsoft Azure) simplifica a criação e o teste de projetos de código antes de torná-los acessíveis. Pode ser usado com praticamente qualquer linguagem de programação ou tipo de projeto. O Azure Pipelines permite que você crie e teste seu código simultaneamente enquanto o distribui para qualquer destino.

Características principais:

- **Facilidade de instalação:** É possível desenvolver, testar e implantar aplicações escritas em Node.js, Python, Java, PHP, Ruby, C/C++, .NET, Android e iOS nos Sistemas Operacionais Linux, macOS e Windows.
- **Centrado no desenvolvedor:** Com facilidade, os usuários podem criar e enviar imagens para registros de contêiner, como o Docker Hub e o Azure Container Registry. Os contêineres podem ser implantados em hosts individuais ou clusters Kubernetes.
- **Suporte de colaboração para DevOps:** Os usuários podem se beneficiar de processos DevOps robustos e vinculados com suporte a contêiner nativo.
- **Flexibilidade de integração:** O Azure Pipelines oferece suporte a um conjunto diversificado de aplicativos de compilação, teste e implantação desenvolvidos pela comunidade, bem como centenas de extensões que vão do Slack ao SonarCloud.
- **Prontidão para empresas:** As empresas podem se beneficiar do encadeamento de compilação simples e das compilações em várias fases que incluem suporte para portais de lançamento, integração de teste, relatórios, etc.

O Azure Pipelines garante que cada projeto de código aberto tenha recursos rápidos de CI/CD. Além disso, os usuários recebem dez tarefas paralelas gratuitas com minutos de compilação ilimitados para qualquer projeto de código aberto.

Preço: custa US$40 por tarefa paralela de CI/CD hospedada pela Microsoft e US$15 por tarefa paralela de CI/CD auto-hospedada adicional com minutos ilimitados.

Pontos positivos: Os recursos são todos sincronizados, tornando-os extremamente fáceis de usar.

Pontos negativos: Pode ser um desafio implementar ajustes específicos do ambiente.

2. GitLab:

Visão geral: GitLab é uma coleção de ferramentas destinadas a serem usadas em vários estágios do ciclo de vida de desenvolvimento de software (SDLC). Em sua essência, é um gerenciador de repositório Git projetado para ser usado na web, com recursos como rastreamento de problemas, relatórios e um wiki online.

Características principais:

- **Facilidade de instalação:** As ferramentas de ramificação permitem que os usuários criem, visualize e gerencie dados e códigos do projeto para simplificar a instalação e o uso.
- **Centrado no desenvolvedor:** Oferece verificação de contêiner, teste de segurança de aplicativo dinâmico (DAST), verificação de dependência e teste de segurança de aplicativo estático (SAST) para dar suporte aos desenvolvedores.
- **Suporte de colaboração para DevOps:** Fornece uma única fonte de escalabilidade e verdade para colaborações de projeto e código.
- **Flexibilidade de integração:** Integração com serviços populares como Jira e Slack estão disponíveis para os usuários.
- **Prontidão para empresas:** Maximiza o retorno geral do desenvolvimento de software para empresas, fornecendo software mais rapidamente e fortalecendo a segurança e a conformidade.

O GitLab é um aplicativo único para todo o ciclo de vida de desenvolvimento de software, incluindo planejamento de projeto e gerenciamento de código-fonte para entrega e manutenção. Isso reduz a necessidade de manter várias ferramentas na pilha.

Preço: o GitLab é gratuito para usuários individuais e custa US$ 19 por usuário de equipe.

Pontos positivos: Gitlab é uma ferramenta de CI/CD extremamente popular, principalmente devido ao seu nível gratuito, que oferece 5 GB de armazenamento gratuito e 10 GB de transferência de dados.

Pontos negativos: Gitlab não fornece serviços de consultoria premium, como integrações em larga escala.

3. Jenkins:

Visão geral: Jenkins é um servidor de código aberto para automação de uso gratuito. No desenvolvimento de software, auxilia na automação dos processos associados à criação, teste e implantação, auxiliando os métodos de integração e entrega contínuas. Nesse caso, é uma solução baseada em servidor que opera em contêineres de servlet, como o servidor web Apache Tomcat.

C**aracterísticas principais:**

- **Facilidade de instalação:** É uma ferramenta baseada em Java compatível vários sistemas operacionais, incluindo

Windows, Linux, macOS e outros sistemas operacionais Unix.
- **Centrado no desenvolvedor:** Oferece maior controle e flexibilidade de configuração aos desenvolvedores, devido à sua interface web.
- **Suporte de colaboração para DevOps:** distribui o trabalho em várias estações de trabalho, permitindo que as equipes concluam compilações, testes e implantações para várias partes interessadas em menos tempo.
- **Flexibilidade de integração:** A funcionalidade do Jenkins pode ser expandida por meio de sua arquitetura de plugins, o que abre as portas para muitas possibilidades para as ferramentas com as quais o Jenkins pode se integrar.
- **Prontidão para empresas:** As empresas podem se beneficiar do Jenkins Enterprise, uma poderosa ferramenta de CI/CD criada com base nos recursos de código aberto do Jenkins.

Apesar de ser uma ferramenta gratuita, ela oferece centenas de plug-ins prontos para uso em seu centro de atualização para reduzir o esforço do desenvolvedor. Além disso, tem uma grande comunidade com eventos e workshops regulares.

Preço: Jenkins é gratuito, embora os usuários devam pagar pelo servidor de hospedagem.

Pontos positivos: Jenkins é fácil de configurar e fornece as maiores bibliotecas de integração entre ferramentas de CI/CD de código aberto.

Pontos negativos: O painel do Jenkins pode ser difícil de usar ao executar vários trabalhos simultaneamente.

O que é DevOps?

DevOps é um conjunto de práticas para automatizar e integrar desenvolvimento e operações. O termo DevOps foi criado a partir da combinação dessas duas palavras: Desenvolvimento (Dev) e operações (Ops). DevOps é a união entre os times de desenvolvimento e operações.

Mas para entender melhor isso, precisamos lembrar como funciona a entrega de software em um modelo convencional. No modelo convencional, a equipe de desenvolvimento desenvolve o produto, gera os binários/executáveis e entrega para o time de infraestrutura (Ops). A equipe de infraestrutura, por sua vez, monta o ambiente para a execução daquele produto e em seguida.

Entrega de software no modelo convencional - Fonte: https://blog.myscrumhalf.com/devops-e-entrega-continua/

Normalmente, nesse modelo vamos encontrar muito retrabalho, ambientes instáveis, erros frequentes, entre outros desafios. Pensando em solucionar esses problemas, surgiu a ideia de colocar as equipes de infraestrutura e desenvolvimento para trabalharem juntas desde a concepção do produto.

Com tudo isso que já falamos aqui, você deve estar imaginando que isso requer muita mudança organizacional e nas equipes. Uma mudança de cultura, de postura e como encarar as coisas.

O movimento DevOps visa remover os silos e promover uma maior colaboração entre esses times.

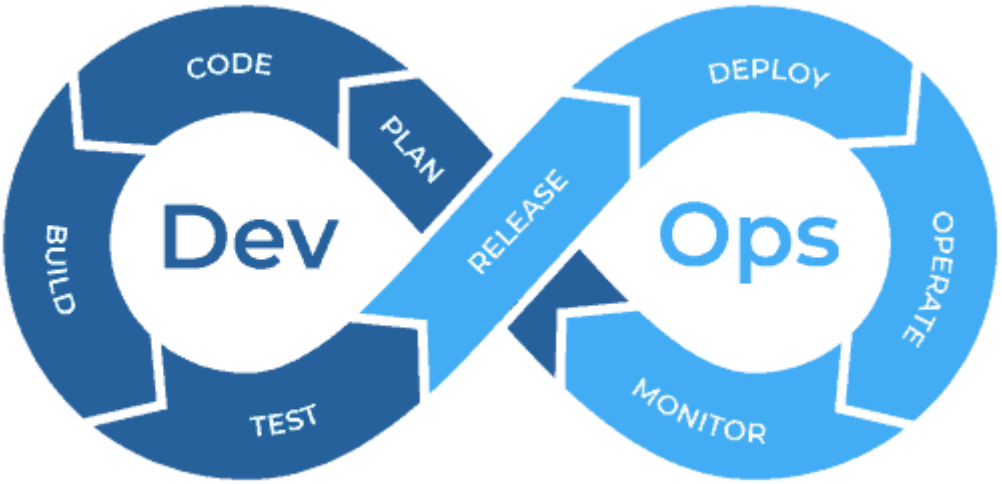


Diagrama DevOps - Fonte: https://blog.4linux.com.br/devsecops-implementacao-em-6-passos/

A responsabilidade compartilhada é um aspecto fundamental do DevOps, não existe DevOps sem responsabilidade compartilhada. Os times compartilham as dores e o sucesso, como em um casamento.

Para exemplificar isso que estamos falando, é muito mais fácil para um time de desenvolvimento perder o interesse na manutenção, sustentação do sistema se essa responsabilidade vai só para o time de operações. Mas, se você torna o time de desenvolvimento responsável por essa parte, você compartilha com ele a dor que as operações sentem. Dessa forma, eles podem pensar juntos em soluções já na concepção do produto ou em automações que resolvam esse ou aquele problema. De forma similar, se você torna o time de operações responsável pelas metas, pelo sucesso do produto, eles trabalham mais perto dos desenvolvedores, e dessa forma, eles podem entender melhor as necessidades operacionais do sistema e ajudar a atendê-las desde o começo.

Mas na prática, como isso funciona?

Tudo começa, com a conscientização dos times. Os desenvolvedores começam a entender sobre monitoramento e sustentação de ambiente e o time de operações começando a entender mais as ferramentas de automação.

Parte desse processo de trazer a cultura DevOps é explicar para seu time que a ideia é a de derrubar a barreira entre as equipes, ao ponto de não conseguirmos identificar onde termina o desenvolvimento e começa operações. A ideia é que eles trabalhem tão juntos que, eventualmente, não haja distinção entre o profissional de desenvolvimento e o profissional de operações.

As etapas do ciclo DevOps

Plan

O estágio de planejamento abrange tudo o que acontece antes dos desenvolvedores começarem a escrever o código, essa é a etapa onde atuam os gerentes de produto ou gerentes de projeto. Requisitos e feedback são coletados de partes interessadas e clientes e usados para construir um roteiro de produto para orientar o desenvolvimento futuro. O roteiro do produto pode ser registrado e rastreado usando um sistema de gerenciamento de tickets, como Jira, Azure DevOps ou Asana, que fornece uma variedade de ferramentas que ajudam a acompanhar o progresso, os problemas e os marcos do projeto.

O roadmap do produto pode ser dividido em Epics, Features e User Stories, criando um backlog de tarefas que levam diretamente aos requisitos dos clientes. As tarefas no backlog podem ser usadas para planejar sprints e alocar tarefas à equipe para iniciar o desenvolvimento.

Code

Depois que a equipe pegou seus cafés e se levantou pela manhã, os desenvolvimentos podem começar a funcionar. Além do kit de ferramentas padrão de um desenvolvedor de software, a equipe tem um conjunto padrão de plug-ins instalados em seus ambientes de desenvolvimento para auxiliar no processo de desenvolvimento, ajudar a impor estilos de código consistentes e evitar falhas de segurança comuns e antipadrões de código.

Isso ajuda a ensinar aos desenvolvedores boas práticas de codificação enquanto auxilia a colaboração, fornecendo alguma consistência à base de código. Essas ferramentas também ajudam a resolver problemas que podem falhar em testes posteriores no pipeline, resultando em menos complicações com falha.

Build

A fase de construção é onde o DevOps realmente entra em ação. Depois que um desenvolvedor conclui uma tarefa, ele envia seu código para um repositório de código compartilhado. Há muitas maneiras de fazer isso, mas normalmente o desenvolvedor envia uma solicitação pull - uma solicitação para mesclar seu novo código com a base de código compartilhada. Outro desenvolvedor revisa as alterações feitas e, uma vez satisfeito, não há problemas, aprova a solicitação pull. Essa revisão manual deve ser rápida e leve, mas é eficaz na identificação de problemas com antecedência.

Simultaneamente, o pull request aciona um processo automatizado que constrói a base de código e executa uma série de testes de unidade,

integração e de ponta a ponta para identificar quaisquer regressões. Se a compilação falhar ou qualquer um dos testes falhar, a solicitação pull falhará e o desenvolvedor será notificado para resolver o problema. Ao verificar continuamente as alterações de código em um repositório compartilhado e executar compilações e testes, podemos minimizar os problemas de integração que surgem ao trabalhar em uma base de código compartilhada e destacar bugs no início do ciclo de vida do desenvolvimento.

Test

Quando uma compilação é bem-sucedida, ela é implantada automaticamente em um ambiente de teste para testes mais profundos e fora de banda. O ambiente de teste pode ser um serviço de hospedagem existente ou pode ser um novo ambiente provisionado como parte do processo de implantação. Essa prática de provisionar automaticamente um novo ambiente no momento da implantação é chamada de infraestrutura como código (IaC) e é uma parte essencial de muitos pipelines de DevOps. Mais sobre isso em um artigo posterior.

Depois que o aplicativo é implantado no ambiente de teste, uma série de testes manuais e automatizados são realizados. O teste manual pode ser o tradicional teste de aceitação do usuário (UAT), em que as pessoas usam o aplicativo como o cliente faria para destacar quaisquer problemas ou refinamentos que devem ser abordados antes da implantação na produção.

Ao mesmo tempo, testes automatizados podem executar varreduras de segurança no aplicativo, verificar alterações na infraestrutura e conformidade com as práticas recomendadas de proteção, testar o desempenho do aplicativo ou executar testes de carga. O teste que é realizado durante esta fase fica a critério da organização e do que é relevante para o aplicativo, mas esse estágio pode ser considerado um banco de testes que permite conectar novos testes sem interromper o fluxo de desenvolvedores ou impactar o ambiente de produção.

Release

A fase de lançamento é um marco em um pipeline de DevOps — é o ponto em que dizemos que uma compilação está pronta para implantação no ambiente de produção. Nesse estágio, cada alteração de código passou por uma série de testes manuais e automatizados, e a equipe de operações pode ter certeza de que problemas de interrupção e regressões são improváveis.

Dependendo da maturidade de DevOps de uma organização, eles podem optar por implantar automaticamente qualquer compilação que chegue a esse estágio do pipeline. Os desenvolvedores podem usar sinalizadores de recursos para desativar novos recursos para que não sejam vistos pelos clientes até que estejam prontos para ação. Esse modelo é considerado o

nirvana do DevOps e é como as organizações conseguem implantar várias versões de seus produtos todos os dias.

Como alternativa, uma organização pode querer ter controle sobre quando as compilações são liberadas para produção. Eles podem querer ter um cronograma de lançamento regular ou apenas lançar novos recursos quando um marco for atingido. Você pode adicionar um processo de aprovação manual no estágio de liberação que permite apenas que determinadas pessoas dentro de uma organização autorizem uma liberação para produção.

As ferramentas permitem que você personalize isso, cabe a você como deseja fazer as coisas.

Deploy

Finalmente, uma compilação está pronta para o grande momento e é lançada em produção. Existem várias ferramentas e processos que podem automatizar o processo de lançamento para tornar os lançamentos confiáveis sem janela de interrupção.

A mesma infraestrutura como código que criou o ambiente de teste pode ser configurada para criar o ambiente de produção. Já sabemos que o ambiente de teste foi construído com sucesso, então podemos ter certeza de que a versão de produção será lançada sem problemas.

Uma implantação bem sucedida nos permite mudar para o novo ambiente de produção sem interrupção. Em seguida, o novo ambiente é construído, ele fica ao lado do ambiente de produção existente. Quando o novo ambiente estiver pronto, o serviço de hospedagem apontará todas as novas solicitações para o novo ambiente. Se, a qualquer momento, for encontrado um problema com a nova compilação, você pode simplesmente dizer ao serviço de hospedagem para apontar as solicitações de volta ao ambiente antigo enquanto você cria uma correção.

Operate

A nova versão já está ativa e sendo usada pelos clientes. Ótimo trabalho!

A equipe de operações agora está trabalhando duro, certificando-se de que tudo está funcionando sem problemas. Com base na configuração do serviço de hospedagem, o ambiente é dimensionado automaticamente com carga para lidar com picos e vales no número de usuários ativos.

A organização também criou uma maneira de seus clientes fornecer feedback sobre seus serviços, bem como ferramentas que ajudam a coletar e selecionar esse feedback para ajudar a moldar o desenvolvimento futuro do produto. Esse ciclo de feedback é importante – ninguém sabe o que quer mais do que o cliente, e o cliente é a melhor equipe de testes do mundo,

doando muito mais horas para testar o aplicativo do que o pipeline de DevOps jamais poderia. Você precisa capturar essa informação, vale seu peso em ouro.

Monitor

A fase “final” do ciclo DevOps é monitorar o ambiente, isso se baseia no feedback do cliente fornecido na fase de operação, coletando dados e fornecendo análises sobre o comportamento do cliente, desempenho, erros e muito mais.

Também podemos fazer alguma introspecção e monitorar o próprio pipeline de DevOps, monitorando possíveis gargalos no pipeline que estão causando frustração ou impactando a produtividade das equipes de desenvolvimento e operações.

Todas essas informações são então enviadas de volta ao Gerente de Produto e à equipe de desenvolvimento para fechar o ciclo do processo. Seria fácil dizer que é aqui que o loop começa novamente, mas a realidade é que esse processo é contínuo. Não há começo nem fim, apenas a evolução contínua de um produto ao longo de sua vida útil, que só termina quando as pessoas seguem em frente ou não precisam mais dele.

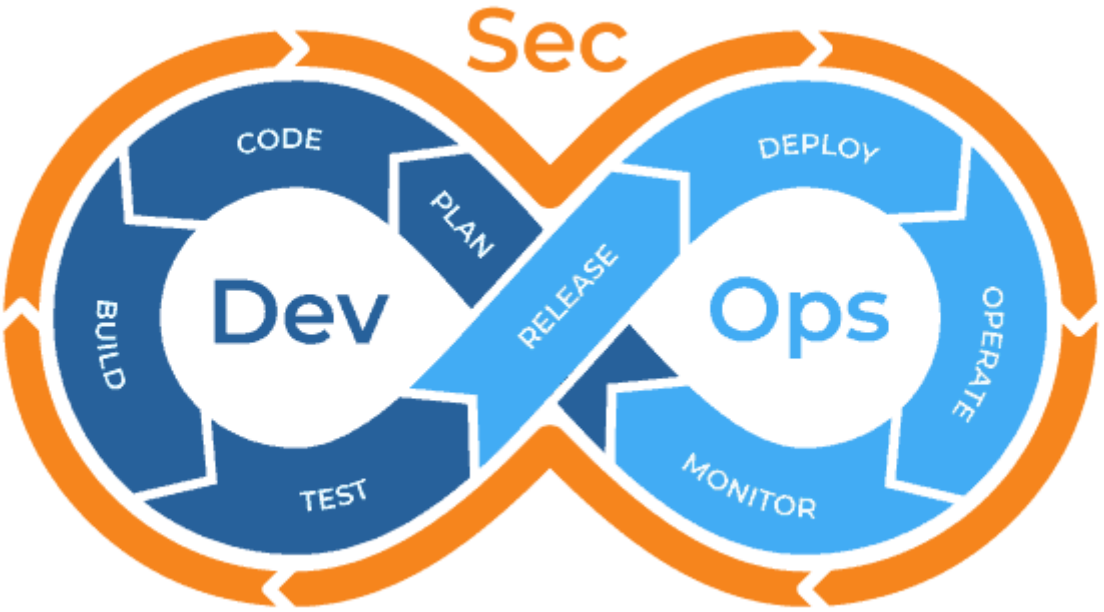
O que é DevSecOps?

A cultura DevOps surgiu para melhorar o cenário de desenvolvimento e entrega de software existentes. Porém, existia um ponto importante que ainda impactava no cenário de desenvolvimento e entrega de software.

DevOps criou uma nova cultura dentro das empresas, com as equipes de desenvolvimento e operações trabalhando juntos desde a concepção do software, foi possível produzir software de qualidade, de forma rápida e possibilitando entregas contínuas em ambiente facilmente escalável. Porém, a equipe de segurança só atuava após o software implementado. Só então eram realizados os testes de segurança, e quando havia alguma solicitação a ser repassada para as equipes de desenvolvimento e/ou operações, todo clico de devops tinha de ser reiniciado.

Ciclo DevOps com segurança atuando após a entrega - Fonte da imagem original: https://blog.4linux.com.br/devsecops-implementacao-em-6-passos/

Como o movimento DevOps visava remover os silos e promover uma maior colaboração entre esses times. E tendo em vista a importância da segurança para o ciclo de desenvolvimento e entrega de software, começou um movimento para trazer a segurança para todo o ciclo, desde a concepção do produto, até a entrega do mesmo.



Ciclo DevSecOps - Fonte: https://blog.4linux.com.br/devsecops-implementacao-em-6-passo

O conceito de DevSecOps baseia-se na automatização dos processos de segurança em uma estrutura de colaboração entre equipes que une a segurança aos processos DevOps desde o início em vez de deixar essa função à cargo de um setor separado de segurança da informação. Uma abordagem DevSecOps integrada promove a redução do risco de segurança sem prejudicar os cronogramas ágeis de desenvolvimento.

O conceito pode dar a impressão da soma de mais processos, mas é importante lembrar que a segurança sempre esteve presente no processo, a mudança visa garantir um desenvolvimento seguro em todas as fases do ciclo de vida de entrega de um software, com integração contínua é essencial para aumentar a qualidade e fazer entregas mais constantes em prazos menores.

Como funciona a cultura DevSecOps?

Para times que buscam integrar conceitos de segurança em sua estrutura DevOps, o processo pode ser feito como uma atualização, usando as ferramentas e os processos DevSecOps certos. Dessa forma, a automação é implementada em todo o pipeline de entrega de software, eliminando erros e reduzindo ataques e momentos de inatividade.

A cultura DevSecOps conta com alguns componentes, entre eles:

- **Análise de código**: Tem o objetivo de entregar o código em pequenos pedaços para identificar vulnerabilidades com maior velocidade.

- **Gerenciamento de mudanças**: Permite que qualquer pessoa envie mudanças para em seguida determinar se a mesma é boa ou ruim, aumentando a velocidade e eficiência do projeto.
- **Monitoramento de conformidade**: Demanda que a equipe esteja pronta para uma auditoria a qualquer momento, promovendo um estado constante de conformidade.
- **Investigação de ameaças**: Identifica ameaças em potencial em cada atualização de código, viabilizando respostas rápidas.
- **Avaliação de vulnerabilidade**: A partir do momento que novas vulnerabilidades são identificadas nas análises de códigos, é analisada a velocidade que as mesmas estão sendo respondidas e corrigidas.
- **Treinamento de segurança**: Engenheiros de software e TI devem ser treinados com as diretrizes para as rotinas do desenvolvimento e operação.

A rotina de testes constante não só leva a um código mais seguro, mas também evita atrasos e imprevistos, distribuindo o trabalho de maneira previsível e consistente por todo o projeto. Através desse processo, as organizações podem cumprir melhor seus prazos, promovendo maior satisfação aos clientes e usuários finais.

Por dentro de uma pipeline DevSecOps

Existem diferentes estágios em um pipeline típico de DevOps; um processo SDLC típico inclui fases como planejar, codificar, construir, testar, liberar e implantar. No DevSecOps, verificações de segurança específicas são aplicadas em cada fase.

- **Plan:** execute a análise de segurança e crie um plano de teste para determinar cenários de onde, como e quando os testes serão feitos.
- **Code:** Implante ferramentas de linting e controles Git para proteger senhas e chaves de API.
- **Build:** ao compilar o código para execução, incorpore ferramentas de teste de segurança de aplicativo estático (SAST) para rastrear falhas no código antes de implantar na produção. Essas ferramentas são específicas para linguagens de programação.
- **Test:** Use ferramentas de teste de segurança de aplicativo dinâmico (DAST) para testar seu aplicativo durante o tempo de execução. Essas ferramentas podem detectar erros associados à autenticação do usuário, autorização, injeção de SQL e endpoints relacionados à API.
- **Release:** Pouco antes de liberar o aplicativo, empregue ferramentas de análise de segurança para realizar testes de penetração completos e verificação de vulnerabilidades.

- **Deploy:** depois de concluir os testes acima em tempo de execução, envie uma compilação segura para produção para implantação final.

Ferramentas no ciclo DevOps/DevSecOps

Existem várias ferramentas open source e comerciais que nos auxiliam na implementação das fases do ciclo de DevSecOps. No vídeo a seguir, vamos conhecer um pouco mais sobre algumas dessas ferramentas:

Aqui vamos falar um pouco sobre algumas dessas ferramentas. Para facilitar, vamos separar as ferramentas por fase.

- **Plan**
    - Microsoft Project
    - Jira
    - Trello
- **Code –** As ferramentas dependem diretamente da linguagem de código adotada para o projeto.
- **Build –** Análoga à fase anterior, as ferramentas dependem diretamente da linguagem de código adotada para o projeto.
- **Test –** A escolha das ferramentas esta diretamente ligada ao tipo de teste que se deseja realizar. Como exemplos temos:
    - Para testes de qualidade do código:
        - SonarQube
    - Para testes funcionais:
        - Selenium
    - Para testes de segurança de código:
        - Veracode
        - Fortify
        - Checkmarx
- **Release**
    - Veracode
    - Fortify
- **Deploy and Operate**
    - AWS
    - Kubernetes (Docker)
    - Ansible
- **Monitor**
    - Nagios
    - New Relics

- **Integração** – Apesar de não ser uma fase, é importante apresentarmos algumas ferramentas que fornecem a base para a pipeline de integração das fases e ferramentas citadas acima.
    - Azure DevOps
    - Jenkins
    - Gitlab CI

Infraestrutura

A infraestrutura é um dos princípios centrais de um processo de desenvolvimento de software – ela é diretamente responsável pela operação estável de um aplicativo de software. Essa infraestrutura pode variar de servidores, balanceadores de carga, firewalls e bancos de dados até clusters de contêiner complexos.

As considerações de infraestrutura são válidas além dos ambientes de produção, pois se espalham por todo o processo de desenvolvimento. Eles incluem ferramentas e plataformas como plataformas CI/CD, ambientes de teste e ferramentas de teste. Essas considerações de infraestrutura aumentam à medida que o nível de complexidade do produto de software aumenta. Muito rapidamente, a abordagem tradicional para gerenciar manualmente a infraestrutura torna-se uma solução não escalável para atender às demandas dos modernos ciclos de desenvolvimento de software rápidos baseados em DevOps.

E é assim que a infraestrutura como código (IaC) tornou-se a solução de infraestrutura nos ambientes atuais de desenvolvimento de software. O IaC permite que você atenda às crescentes necessidades de mudanças de infraestrutura de maneira escalável e rastreável.

O que é IaC?

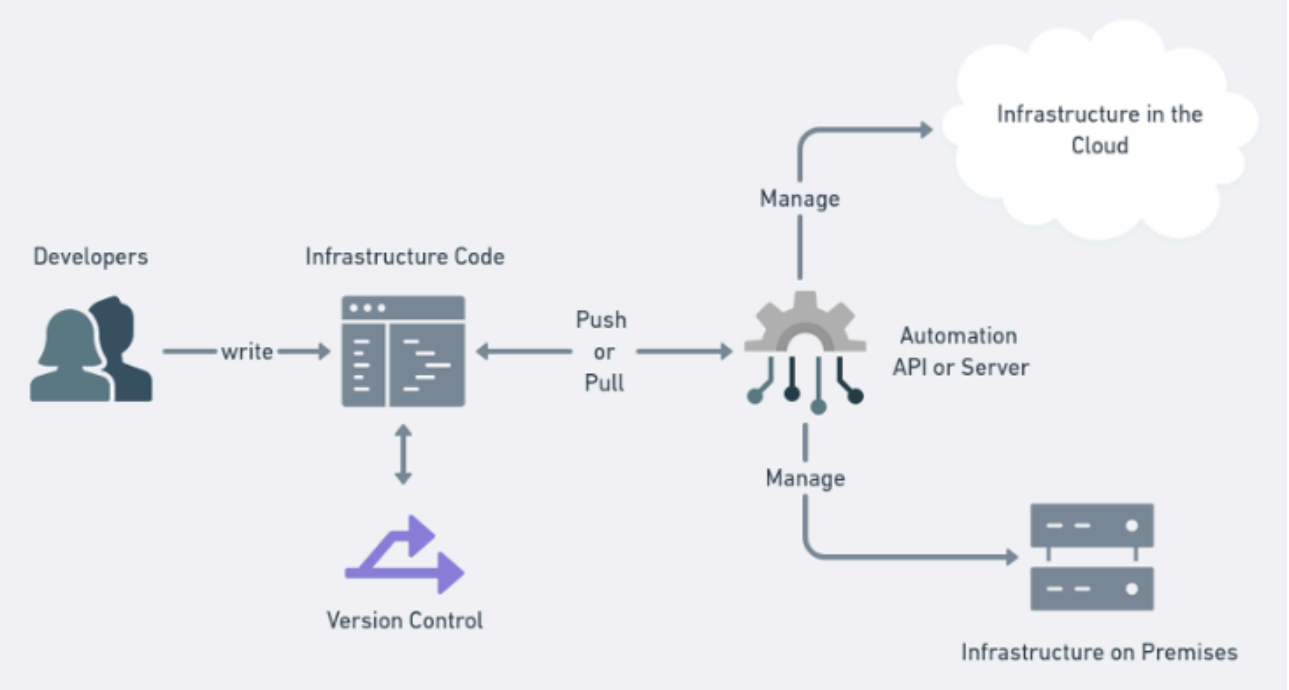
IaC - Infrastructure as Code (ou Infraestrutura como código) é o gerenciamento da infraestrutura (redes, máquinas virtuais, balanceadores de carga e topologia de conexão) em um modelo descritivo, usando o mesmo controle de versão que a equipe de DevOps usa para o código-fonte. Seguindo o princípio de que o mesmo código-fonte gera o mesmo binário, um modelo de IaC gera o mesmo ambiente toda vez que é aplicado. O IaC é uma prática fundamental do DevOps usada em conjunto com a entrega contínua.

Como a infraestrutura é definida como código, ela permite que os usuários editem e distribuam facilmente as configurações, garantindo o estado desejado da infraestrutura. Isso significa que você pode criar configurações de infraestrutura reproduzíveis.

Além disso, definir infraestrutura como código também:

- Permite que a infraestrutura seja facilmente integrada aos mecanismos de controle de versão para criar alterações de infraestrutura rastreáveis e auditáveis.
- Fornece a capacidade de introduzir automação extensiva para gerenciamento de infraestrutura. Tudo isso faz com que o IaC seja integrado aos pipelines de CI/CD como parte integrante do SDLC (Ciclo de Vida no Desenvolvimento de Software).

- Elimina a necessidade de provisionamento e gerenciamento manual de infraestrutura. Assim, ele permite que os usuários gerencie facilmente o inevitável desvio de configuração da infraestrutura e configurações subjacentes e mantenham todos os ambientes dentro da configuração definida



Fonte: https://forum.huawei.com/enterprise/en/get-to-know-about-infrastructure-as-code/thread/864719-893

Quando e como usar a infraestrutura como código

O IaC pode parecer desnecessário para requisitos de infraestrutura mais simples e menos complexos, mas isso não é preciso. Qualquer pipeline de desenvolvimento de software moderno deve usar a infraestrutura como código para lidar com a infraestrutura.

Além disso, as vantagens do IaC superam em muito quaisquer despesas gerais de implementação e gerenciamento.

Vantagens do IAC

Aqui estão os principais benefícios do IaC:

- Reduz o shadow IT dentro das organizações e permite mudanças de infraestrutura oportunas e eficientes que são feitas em paralelo ao desenvolvimento de aplicativos.
- Integração direta com plataformas CI/CD.
- Habilita alterações de configuração e infraestrutura controladas por versão, levando a configurações rastreáveis e auditáveis.
- Padroniza facilmente a infraestrutura com configurações reproduzíveis.

- Gerencia de forma eficaz o desvio de configuração e mantém a infraestrutura e as configurações em seu estado desejado.
- Ter a capacidade de dimensionar facilmente o gerenciamento de infraestrutura sem aumentar o CapEx (despesas de capital) ou o OpEx (despesa operacional). Com o IaC, você reduzirá os gastos gerais de CapEx e OpEx, pois a automação elimina a necessidade de interações manuais demoradas e reduz configurações incorretas.

Quando usar IAC

Não tem certeza de quando usar o IAC? A resposta mais simples é sempre que você precisar gerenciar qualquer tipo de infraestrutura.

No entanto, torna-se mais complexo com os requisitos e ferramentas exatos. Alguns podem exigir gerenciamento de infraestrutura rigoroso, enquanto outros podem exigir gerenciamento de infraestrutura e configuração. Em seguida, vêm questões específicas da plataforma, como se a ferramenta possui o conjunto de recursos necessários, implicações de segurança, integrações etc. Além disso, a curva de aprendizado entra em jogo, pois os usuários preferem uma ferramenta mais simples e direta do que uma complexa.

Mudanças de infraestrutura rápidas e rastreáveis

A infraestrutura como código tornou-se uma parte vital do desenvolvimento de aplicativos modernos e dos pipelines de implantação. Isso é alcançado ao facilitar mudanças de infraestrutura rápidas e rastreáveis que se integram diretamente às plataformas de CI/CD. A infraestrutura como código é crucial para:

- Facilitando o gerenciamento de infraestrutura escalável
- Gerenciando com eficiência o desvio de configuração em todos os ambientes

Começar com a infraestrutura como código pode parecer assustador com muitas ferramentas e plataformas diferentes direcionadas a diferentes casos de uso. No entanto, supere esse obstáculo e você terá um poderoso mecanismo de gerenciamento de infraestrutura ao seu alcance.

Ferramentas de gerenciamento de configuração

O provisionamento de infraestrutura sempre foi um processo manual, caro e demorado. Porém, agora, o gerenciamento de infraestrutura migrou do hardware físico em data centers, apesar de ainda ser um componente da sua organização, para virtualização, containers e cloud computing.

O número de componentes de infraestrutura aumentou com a cloud computing, já que mais aplicações são colocadas em produção todos os dias e as suas infraestruturas precisam ser flexíveis para as constantes alterações, escalas e desativações. Nos dias atuais, não ter a implementação de uma prática IAC e gerenciar a escala da infraestrutura, fica cada vez mais complicado.

O IAC pode ajudar na organização do gerenciamento de necessidades de infraestrutura de TI, otimizando a consistência e diminuindo erros e a necessidade de configuração manual.

Abaixo, estão listadas as principais ferramentas de gerenciamento de configuração e automação de servidor, que podem ser usadas para atingir a IAC.

- Chef
- Puppet
- Red Hat Ansible Automation Platform
- Saltstack
- Terraform
- AWS CloudFormation

Para que serve a infrastructure as code nas práticas de DevOps?

Qual é a importância do DevOps infraestrutura como código? A IAC é uma parte essencial da implementação de práticas de DevOps e de integração e entrega contínuas (CI/CD). Ela alivia as funcionalidades realizadas pelos desenvolvedores, já que elimina a maior parte do trabalho de provisionamento. Dessa maneira, eles podem executar um script para preparar a infraestrutura.

Assim, as implantações de aplicações não ficam esperando pela infraestrutura e os administradores do sistema não precisam gerenciar processos de maneira manual e desperdiçando um longo tempo. A prática de CI/CD conta com o monitoramento e automação a longo prazo durante o ciclo de vida da aplicação, desde a junção e o teste até a entrega e a implantação.

Utilizando uma abordagem de DevOps, o alinhamento das equipes de desenvolvimento e de operações produzem menos erros, menos implantações manuais e menos inconsistências. O IAC ajuda a alinhar essas equipes de operação e desenvolvimento, já que ambas podem usar a mesma descrição da implantação de aplicações, compatível com a abordagem de DevOps.

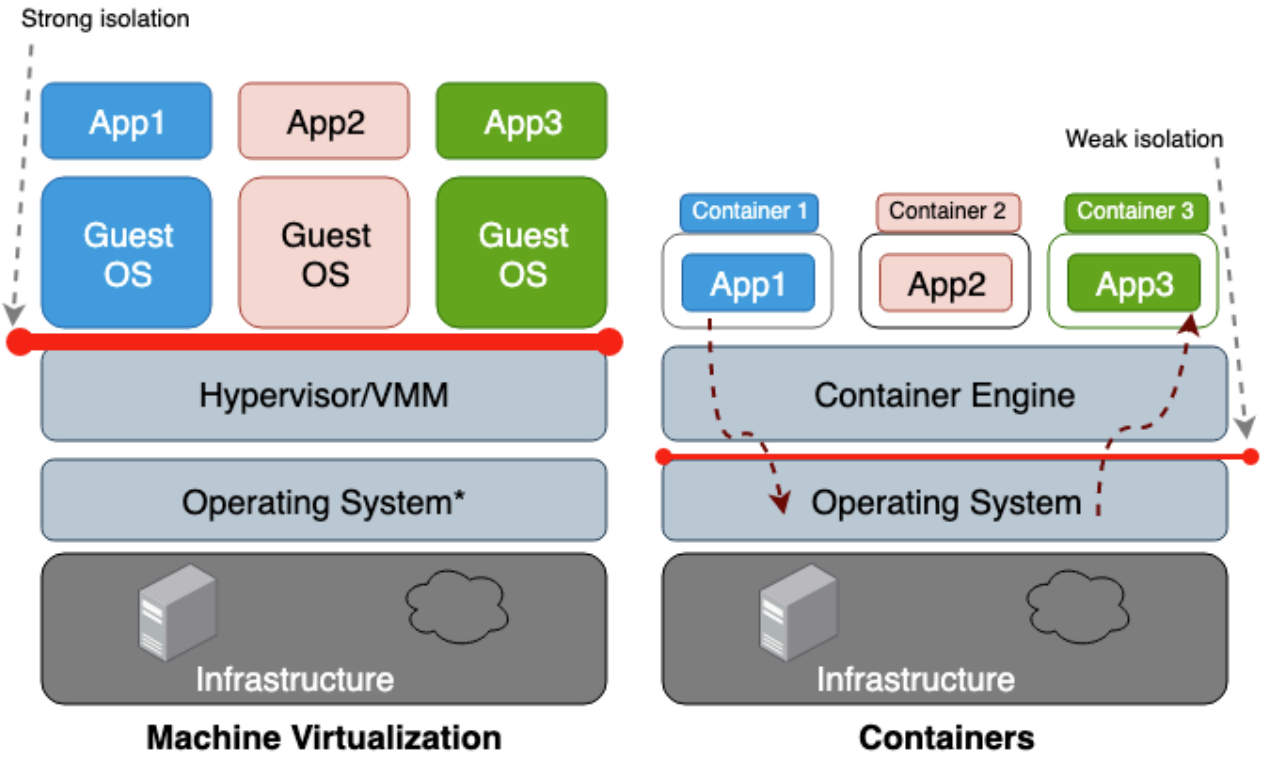
Containers

O que são Containers?

Os containers são uma forma de virtualização do sistema operacional. Um único contêiner pode ser usado para executar qualquer coisa, desde um pequeno microsserviço ou processo de software até um aplicativo maior. Dentro de um container estão todos os executáveis necessários, código binário, bibliotecas e arquivos de configuração. Em comparação com as abordagens de virtualização de servidor ou máquina, no entanto, os containers não contêm imagens do sistema operacional. Isso os torna mais leves e portáteis, com uma sobrecarga significativamente menor. Em implantações de aplicativos maiores, vários containers podem ser implantados como um ou mais clusters de container. Esses clusters podem ser gerenciados por um orquestrador de container, como o Kubernetes.

Containers vs Virtual Machines (VM)?

Containers e máquinas virtuais (VM) são tecnologias de virtualização de recursos muito semelhantes. A virtualização é o processo no qual um recurso singular do sistema, como RAM, CPU, disco ou rede, pode ser "virtualizado" é representado como vários recursos. O principal diferencial entre containers e máquinas virtuais é que as máquinas virtuais virtualizam uma máquina inteira até as camadas de hardware e os containers apenas virtualizam as camadas de software acima do nível do sistema operacional.



Fonte: https://unit42.paloaltonetworks.com/making-containers-more-isolated-an-overview-of-sandboxed-container-technologies/

Contêiner

Os contêineres são pacotes de software leves que contêm todas as dependências necessárias para executar o software. Essas dependências incluem coisas como bibliotecas do sistema, pacotes de código de terceiros externos e outros aplicativos no nível do sistema operacional. As dependências incluídas em um contêiner existem em níveis de pilha superiores ao sistema operacional.

Prós

- **Velocidade de iteração:** Como os contêineres são leves e incluem apenas software de alto nível, eles são muito rápidos para modificar e iterar.
- **Ecossistema robusto:** A maioria dos sistemas de tempo de execução de contêiner oferece um repositório público hospedado de contêineres pré-fabricados. Esses repositórios de contêiner contêm muitos aplicativos de software populares, como bancos de dados ou sistemas de mensagens, e podem ser baixados e executados instantaneamente, economizando tempo para as equipes de desenvolvimento

Contras

- **Explorações de host compartilhado:** Todos os contêineres compartilham o mesmo sistema de hardware subjacente abaixo da camada do sistema operacional, é possível que uma exploração em um contêiner possa sair do contêiner e afetar o hardware compartilhado. Os tempos de execução de contêiner mais populares têm repositórios públicos de contêineres pré-criados. Existe um risco de segurança ao usar uma dessas imagens públicas, pois elas podem conter explorações ou podem ser vulneráveis a serem invadidas por agentes nefastos.

Principais provedores de contêineres

- **Docker:** O Docker é o contêiner mais popular e amplamente utilizado. O Docker Hub é um repositório público gigante de aplicativos de software em contêineres populares. Os contêineres no Docker Hub podem ser baixados e implantados instantaneamente em um tempo de execução local do Docker.
- **RKT:** Pronunciado "Rocket", o RKT é um sistema de contêineres focado em segurança. Os contêineres RKT não permitem a funcionalidade de contêiner inseguro, a menos que o usuário habilite explicitamente recursos inseguros. Os contêineres RKT

visam resolver os problemas de segurança exploratórios de contaminação cruzada subjacentes que outros sistemas de tempo de execução de contêineres sofrem.
- **Contêineres Linux (LXC):** O projeto Linux Containers é um sistema de tempo de execução de contêiner Linux de código aberto. O LXC é usado para isolar processos operacionais em nível de sistema uns dos outros. Na verdade, o Docker usa o LXC nos bastidores. Os contêineres do Linux visam oferecer um tempo de execução de contêiner de código aberto neutro para fornecedores.
- **CRI-O:** CRI-O é uma implementação do Kubernetes Container Runtime Interface (CRI) que permite o uso de runtimes compatíveis com Open Container Initiative (OCI). É uma alternativa leve ao uso do Docker como o tempo de execução do Kubernetes.

Máquina Virtual

Máquinas virtuais são pacotes de software que fornecem emulação completa de dispositivos de hardware de baixo nível, como dispositivos de CPU, disco e rede. As máquinas virtuais também podem incluir uma pilha de software complementar para executar no hardware emulado. Esses pacotes de hardware e software combinados produzem um snapshot (uma espécie de cópia) totalmente funcional de um sistema computacional.

Prós

- **Segurança de isolamento total:** As máquinas virtuais são executadas isoladamente como um sistema totalmente autônomo. Isso significa que as máquinas virtuais são imunes a qualquer exploração ou interferência de outras máquinas virtuais em um host compartilhado. Uma máquina virtual individual ainda pode ser sequestrada por uma exploração, mas a máquina virtual explorada será isolada e incapaz de contaminar outras máquinas virtuais vizinhas.
- **Desenvolvimento interativo:** Os contêineres geralmente são definições estáticas das dependências e configurações esperadas necessárias para executar o contêiner. As máquinas virtuais são mais dinâmicas e podem ser desenvolvidas interativamente. Depois que a definição básica de hardware é especificada para uma máquina virtual, a máquina virtual pode ser tratada como um computador básico. O software pode ser instalado manualmente na máquina virtual e a máquina virtual pode ser capturada para capturar o estado de configuração atual. Os instantâneos da máquina virtual podem ser usados para restaurar a máquina virtual para esse ponto no tempo ou ativar máquinas virtuais adicionais com essa configuração.

Contras

- **Velocidade de iteração:** As máquinas virtuais são demoradas para construir e regenerar porque abrangem um sistema de pilha completa. Quaisquer modificações em um instantâneo de máquina virtual podem levar um tempo significativo para regenerar e validar se se comportam conforme o esperado.
- **Custo do tamanho do armazenamento:** As máquinas virtuais podem ocupar muito espaço de armazenamento. Eles podem crescer rapidamente para vários gigabytes de tamanho. Isso pode levar a problemas de falta de espaço em disco na máquina host das máquinas virtuais.

Principais provedores de VMs

- **VirtualBox:** O Virtualbox é um sistema de emulação de arquitetura x86 gratuito e de código aberto de propriedade da Oracle. O Virtualbox é uma das plataformas de máquinas virtuais mais populares e estabelecidas, com um ecossistema de ferramentas complementares para ajudar a desenvolver e distribuir imagens de máquinas virtuais.
- **VMware:** A VMware é uma empresa de capital aberto que construiu seus negócios em uma das primeiras tecnologias de virtualização de hardware x86. VMware vem incluído com um hypervisor que é um utilitário que irá implantar e gerenciar várias máquinas virtuais. VMware tem UI robusta para gerenciar máquinas virtuais. VMware é uma ótima opção de máquina virtual corporativa que oferece suporte.
- **QEMU:** QEMU é a opção de máquina virtual de emulação de hardware mais robusta. Tem suporte para qualquer arquitetura de hardware genérica. O QEMU é um utilitário somente de linha de comando e não oferece uma interface gráfica de usuário para configuração ou execução. Essa compensação torna o QEMU uma das opções de máquina virtual mais rápidas.

Qualidade de software

O que é qualidade de software?

Para definir qualidade de software podemos afirmar que a qualidade de software é uma área de conhecimento em engenharia de software e seu objetivo é garantir a qualidade do software através da definição e normatização de processos de desenvolvimento.

O conceito de qualidade é muito subjetivo, pois está relacionado diretamente com a percepção de cada indivíduo. A qualidade é feita de características percebidas, não só para o usuário final, quanto também pela própria equipe de desenvolvimento.

Falando de um produto ou serviço podemos dizer que qualidade é quando estes estão atendendo as exigências dos clientes, tem uma boa relação custo/benefício, sua utilização é de fácil adequação e tem um grande valor agregado.

Não adianta termos um software que funciona, atende as expectativas do cliente, mas a equipe encontra dificuldades para mantê-lo.

Então é importante quando falamos de qualidade de software sempre pensar por esses dois ângulos.

Devido a estas diferentes visões, a qualidade de software mostra a sua grande importância no processo de desenvolvimento, pois na verdade sua função é garantir que o software que será entregue ao cliente no fim do projeto, atenderá a todas as suas expectativas e que mesmo sem saber, o cliente receberá um produto desenvolvido com a utilização de boas práticas e técnicas de desenvolvimento adequadas.

Quais insights o teste de qualidade de software oferece?

Adotar técnicas de teste de software permite a padronização e verificação do desempenho de um sistema. Realizar testes de software evita transtornos, atrasos em entregas e manutenções em excesso, pois:

- Indica se seu software está cumprindo seu papel de forma efetiva;
- Identifica eventuais correções e/ou otimizações;
- Verifica a velocidade de resposta e eficiência do programa;
- Aponta se o software atende aos padrões e normas de qualidade.

Para alcançar estes resultados, é importante utilizar ferramentas de automação de testes. Elas facilitam a detecção de problemas, já que em

muitos momentos precisam ser aplicadas várias vezes seguidas, até a falha ser corrigida.

Métricas de avaliação

- **Alcance: as funções escolhidas satisfazem as necessidades do usuário final do produto?**

A métrica de alcance está diretamente relacionada ao público que ela atinge. Assim, é possível pensar em idiomas, plataformas onde é possível utilizar seu software e acessibilidade.

- **Usabilidade: quão fácil é utilizar este software?**

Usabilidade está ligada à facilidade com que o usuário irá operar sua aplicação e à intuitividade. Ela é **importante para garantir uma boa experiência**.

- **Profundidade: qual o nível das ramificações de atuação do software?**

Esta métrica está relacionada à arquitetura de software, ao mapa de operações e sua complexidade. Para mensurá-la, é preciso levar em consideração a interface, banco de dados e análise destes dados.

- **Portabilidade: é possível integrar o sistema a outras diversas plataformas com facilidade?**

É a capacidade de uso do programa em diferentes plataformas e diferentes condições. Ou seja, é também a **possibilidade de integração com outras ferramentas**.

- **Confiabilidade: o desempenho do software é mantido ao longo do tempo?**

A confiabilidade é a menor taxa de falhas possível, fazendo com que o software rode sem erros.

- **Manutenibilidade: qual a dificuldade para realizar correções, atualizações e alterações?**

Esta métrica está relacionada à **facilidade de um software de passar por manutenções e atualizações**. É necessário que, após estas ações, o software não apresente erros.

- **Eficiência: os recursos e o tempo aplicados são compatíveis com o desempenho esperado?**

A eficiência de um software se refere ao tempo de resposta para executar uma função. Esta métrica também é importante para oferecer uma boa experiência ao usuário.

Realizar a avaliação de qualidade de software é essencial para alcançar notoriedade com o seu produto e aumentar a competitividade.

Qualidade de Software para DevOps

Como vimos acima, para garantir a qualidade de um software existem diversas técnicas, modelos de controle e garantia de qualidade que podem ser adotadas. Uma das principais formas de garantirmos a qualidade do software dentro do ciclo DevOps/DevSecOps, é através da utilização de testes de software.

A seguir, seguem alguns dos testes mais utilizados em fluxos DevOps/DevSecOps:

- **Testes unitários:** Funções testadas isoladamente. São testes mais rápidos e mais confiáveis já que o escopo é bem reduzido.
- **Testes de integração:** Combina componentes para garantir que a comunicação entre eles atende aos requisitos estabelecidos.
- **Testes de contrato:** Validar se o contrato estabelecido entre um consumidor e o provedor está sendo respeitado.
- **Testes End-to-End:** Responsável por validar os fluxos, literalmente, de uma ponta a outra para garantir que todo o ambiente atende os requisitos.
- **Testes de Regressão Visual:** testes que expõem qualquer alteração visual em comparação com uma baseline.
- **Testes de Sistema:** Testa o sistema já completamente integrado e pode ter como objetivo validar os requisitos funcionais e não funcionais.
- **Testes de Mutação:** Altera o código em tempo de execução e espera que os testes falhem, mostrando se os mesmos são efetivos.
- **Testes de Performance:** Avaliar a capacidade de resposta, robustez, disponibilidade, confiabilidade e escalabilidade.
- **Testes de Chaos:** Introduzir falhas na sua infraestrutura para observar como ela se recupera e antecipar possíveis problemas.

Security by Design

O que é?

Um dos pilares de DevSecOps é incorporar boas práticas de segurança da informação para diminuir as vulnerabilidades desde o início do desenvolvimento de software, até a entrega do software.

Security by Design nasceu para desempenhar esse papel, incorporando boas práticas de segurança desde o planejamento do software para diminuir as vulnerabilidades.

Security by Design propõe uma relação de trabalho positiva entre as equipes de desenvolvimento e segurança, com requisitos claros e apropriados, além da possibilidade de testar a segurança do código-fonte buscando uma adequação incorporada ao desenvolvimento do software desde seu planejamento inicial e concepção.

Desenvolvimento Seguro e Security by Design

O desenvolvimento seguro surgiu como uma resposta ao crescimento das vulnerabilidades de segurança nos sistemas. Consiste em acrescentar atividades de segurança no ciclo de desenvolvimento de um software. Apresenta uma abordagem estruturada e flexível para implementação imediata, porém muitas equipes de desenvolvimento possuem dificuldades na sua aplicação, principalmente pela falta de conhecimento nas técnicas de segurança.

As equipes se deparam constantemente com a necessidade de gerenciar prazos, novas tecnologias, usabilidade, desempenho e segurança dos sistemas ao mesmo tempo, e sabem que nem sempre é possível atender todos esses requisitos.

A integração dos requisitos de segurança ao trabalho diário dos desenvolvedores é uma abordagem favorável para que as equipes desenvolvam sistemas mais seguros. É necessário compreender que a segurança é responsabilidade de todos.

Com o surgimento de DevSecOps, vimos a chegado do conceito de *Shift Left,* é mais eficiente e menos custoso incluir preocupações com segurança e privacidade desde o início do desenvolvimento, reduzindo o volume de vulnerabilidades, retrabalho e, consequentemente, o custo da correção de uma falha no ambiente de produção. Por isso, muitas organizações passaram a incluir um recurso chamado de Security Champion.

 nos times de desenvolvimento, o qual tem a responsabilidade por olhar a segurança durante a esteira de desenvolvimento.

Benefícios do Security by Design

O conceito de Security by Design descreve as melhores práticas e padrões de segurança aplicados ao design da arquitetura e, em seguida, usados como princípios orientadores para o time de desenvolvimento. Por isso é considerado uma das principais abordagens para garantir a segurança e a privacidade dos sistemas.

Security by Design considera a veracidade das práticas maliciosas e os devidos cuidados são tomados para minimizar o impacto na antecipação de uma vulnerabilidade de segurança. Além disso, uma boa abordagem de Security by Design proporciona:

- Economia: resolver problemas de segurança no início é muito mais eficiente e econômico.
- Resiliência: as práticas de Security by Design resultam em um sistema resiliente, em que a segurança é incorporada por padrão ao invés de adicionada às pressas como uma correção.
- Correção de vulnerabilidades: a implementação de uma variedade de práticas de Security by Design (conscientização, conhecimento, ferramentas e verificações) permite que as falhas de segurança sejam removidas com mais facilidade e rapidez do que testando somente no final do ciclo de desenvolvimento.
- Adaptação: determinar exatamente quais erros foram cometidos permite adaptar o processo de desenvolvimento para evitar erros futuros.

Princípios do Security by Design

A OWASP sugere alguns princípios importantes para o desenvolvimento de software seguro que são aplicados em Security by Design.

1. **Minimizar a superfície de ataque** - é usado para restringir as funções que os usuários têm permissão para acessar, contribuindo com a redução de vulnerabilidades. Com a integração de ferramentas de proteção já existentes, é possível desenvolver um ecossistema de monitoramento e correções em tempo real.
2. **Estabelecimento de padrões -** Padrões de desenvolvimento ajudam a entender as implicações de segurança das práticas de desenvolvimento e implantação de código.
3. **Princípio do menor privilégio -** Sugere fornecer as permissões necessárias para que um usuário realize suas tarefas, com um tempo determinado e os direitos mínimos estabelecidos. O princípio do menor privilégio é uma estratégia que pode ser incorporado ao Security by Design promovendo a segurança das informações e privacidade. A atribuição de permissões a um

usuário pode impedir que ele execute tarefas para as quais não está autorizado, como acessar, obter ou modificar informações.

4. **Princípio da defesa em profundidade -** A defesa em profundidade é um conjunto de práticas que se concentram na proteção, detecção e reação de invasões. Para isso, são usados softwares de segurança e ferramentas para a construção de uma estratégia contra ataques. O uso de ferramentas de segurança como firewalls, antivírus, filtragem de conteúdo, criptografia e controle de acesso colaboram para prevenção de ataques.
5. **Falhar com segurança -** A manipulação segura de erros é um aspecto importante para uma software seguro e para o Security by Design. Existem dois tipos de erros que merecem destaque:
    1. O primeiro são as exceções que ocorrem no processamento de um controle de segurança.
    2. Outro tipo de exceção relevante à segurança está no código que não faz parte de um controle de segurança.
6. **Não confie nos serviços -** Um modelo de confiança zero (conhecido também como Zero Trust) é composto pela recomendação de que as empresas não devem confiar em ninguém ou em nenhum dispositivo ou sistema por padrão e devem verificar todas as conexões antes de permitir o acesso à sua rede. A criação desse modelo foi uma resposta às antigas abordagens de segurança, baseadas na suposição de que a ameaça interna era inexistente e que a segurança da informação deve focar apenas na defesa contra ameaças externas. As ameaças internas são representadas por colaboradores, ex-colaboradores, parceiros de negócio, prestadores de serviços ou qualquer pessoa que tem acesso a informações privilegiadas. As empresas podem levar anos para descobrir a presença dessas ameaças em sua estrutura.
7. **Segregação de funções -** A segregação de funções e responsabilidades é o controle de acesso baseado no papel, na atividade ou na função de um usuário dentro de um sistema. A utilização de um perfil por função (ou RBAC – Role Based Access Control) providencia um modelo para administrar privilégios de acessos aos sistemas e infraestrutura de uma empresa. O perfil por função consegue agrupar os acessos, possibilitando uma visão geral dos privilégios e controlando os acessos de uma forma segura para o Security by Design.
8. **Evitar a segurança por obscuridade -** A segurança por obscuridade é quando os desenvolvedores codificam os sistemas de forma secreta acreditando que ninguém será capaz de encontrar as vulnerabilidades do software. O problema com essa técnica é a dependência em relação ao sigilo da implementação do projeto como forma principal de prover segurança para o sistema. Geralmente, as pessoas que fazem uso dessa técnica assumem que o

não conhecimento das vulnerabilidades de um software é um indicativo de segurança.

9. **Mantenha a segurança simples -** No início de uma implementação de Security by Design é comum fazer uso de ferramentas, processos e controles em favor da segurança de sistemas, mas é necessário refletir sobre a relevância de todos esses controles, eles acrescentam mais segurança ou burocracia aos sistemas? A existência de muitas ferramentas pode aumentar as brechas de segurança em vez de extingui-las, assim como procedimentos pouco documentados ou falta de automações que podem deixar usuários esperando demais por um acesso.
10. **Segurança no processo de manutenção do software -** As vulnerabilidades em sistemas precisam ser estudadas pelo time de desenvolvimento para uma correção eficiente mesmo. É preciso entender o comportamento da vulnerabilidade de forma estrutural no sistema e verificar se existem outros componentes que podem ser afetados pela mesma vulnerabilidade. A falta de um processo ou controle para realizar as correções de problemas pode causar o surgimento de novos problemas e brechas de seguranças nos sistemas. Um processo contínuo de gestão de vulnerabilidades é visto como um aliado para as equipes de desenvolvimento, atuando na identificação, análise, classificação e tratamento das vulnerabilidades. Esse processo busca medir o progresso e avaliar os riscos aos quais os sistemas estão submetidos, colaborando com uma estratégia de Security by Design.

AppSec

O que é?

Application Secutiry (Segurança de aplicação) é uma disciplina de processos, ferramentas e práticas com o objetivo de proteger as aplicações contra ameaças durante todo o ciclo de vida do software (SDLC). Os criminosos cibernéticos são organizados, especializados e motivados a encontrar e explorar vulnerabilidades em aplicações corporativas para roubar dados, propriedade intelectual e informações confidenciais. AppSec pode ajudar as organizações a proteger todos os tipos de aplicações(como legados, desktops, web, móveis, microsserviços) usados por partes interessadas, internas e externas, incluindo clientes, parceiros de negócios e funcionários.

Por que AppSec?

A maioria dos ataques bem-sucedidos têm como alvo vulnerabilidades exploráveis que residem na camada de aplicativo, indicando a necessidade de os departamentos de TI corporativos estarem mais atentos à segurança da aplicação. Para agravar ainda mais o problema, o número e a complexidade das aplicações estão crescendo. Há apenas alguns anos atrás, o desafio de segurança de software era proteger aplicações desktop e sites estáticos que eram bastante inócuos e fáceis de definir e proteger. Hoje em dia, esse cenário é muito mais complexo, considerando o desenvolvimento terceirizado, o número de aplicativos legados, juntamente com o desenvolvimento interno que aproveita componentes de software de terceiros, de código aberto e comerciais.

As organizações precisam de soluções de AppSec que cubram todas as suas aplicações, desde as usadas internamente até as aplicações externas usadas nos telefones celulares dos clientes. Essas soluções devem cobrir todo o estágio de desenvolvimento e oferecer testes após o uso de um aplicativo para monitorar possíveis problemas. As soluções de AppSec devem ser capazes de testar aplicativos da Web em busca de vulnerabilidades potenciais e exploráveis, ter a capacidade de analisar código, ajudar a gerenciar os processos de gerenciamento de segurança e desenvolvimento, coordenar esforços e permitir a colaboração entre as várias partes interessadas. As soluções também devem oferecer testes de segurança que sejam fáceis de usar e implantar.

Testes e Técnicas utilizadas em AppSec

- **SAST -** Static Application Security Testing, verifica os arquivos de código fonte do aplicativo, identifica com precisão a causa raiz e ajuda a corrigir as falhas de segurança subjacentes. Também garante a conformidade com as diretrizes e padrões de codificação sem realmente executar o código subjacente.
- **SCA -** Software composition analysis é um processo automatizado que identifica o software de código aberto em uma base de código. Essa análise é realizada para avaliar a segurança, a conformidade da licença e a qualidade do código.
- **DAST -** Dynamic Application Security Testing, simula ataques controlados em um aplicativo ou serviço da Web em execução para identificar vulnerabilidades exploráveis em um ambiente em execução.
- **IAST -** Interactive Application Security Testing, foi projetado para solucionar as deficiências do SAST e do DAST combinando elementos de ambas as abordagens. Usa instrumentação de software para avaliar o desempenho de um aplicativo e detectar vulnerabilidades. Agentes e sensores são executados para analisar de maneira contínua o funcionamento do aplicativo durante testes automatizados, testes manuais ou uma combinação dos dois.
- **RASP -** Run-time Application Security Protection, um pouco diferente dos anteriores, é menos uma ferramenta de teste e mais uma ferramenta de segurança. Ele está conectado a uma aplicação ou seu ambiente em tempo de execução e pode controlar a execução do aplicativo. O RASP permite que a aplicação execute verificações de segurança contínuas em si mesmo e responda a ataques em tempo real encerrando a sessão de um invasor e alertando os defensores sobre o ataque.
- **OFUSCAÇÂO DE CÓDIGO –** Técnica comumente utilizada pelos hackers para ocultar seus softwares maliciosos, recentemente foi adicionada a ferramentas de segurança para permitirem que o desenvolvedor faça o mesmo processo, porém, com o intuito de proteger seu código contra ataques.
- **FERRAMENTAS DE CRIPTOGRAFIA E ANTI-ADULTERAÇÃO (anti-tampering)** - esses são outros métodos que podem ser usados para impedir que os hackers obtenham informações sobre seu código.
- **FERRAMENTAS DE DETECÇÃO DE AMEAÇAS -** essas ferramentas examinam o ambiente ou a rede em que suas aplicações estão sendo executadas e fazem uma avaliação sobre possíveis ameaças e relações de confiança mal utilizadas. Algumas ferramentas podem fornecer “impressões digitais” do dispositivo para determinar se um telefone celular foi desbloqueado (root) ou comprometido.

OWASP

O que é?

OWASP - Open Web Application Security Project é uma entidade sem fins lucrativos e com reconhecimento internacional, atuando com foco na colaboração para o fortalecimento da segurança de softwares em todo o mundo. Ela é formada por uma comunidade composta de especialistas em segurança espalhados pelo mundo, compartilhando conhecimento e diferentes experiências sobre as vulnerabilidades, ameaças, ataques e contramedidas existentes.

OWASP começou como um projeto voltado para trazer segurança para aplicações web. Atualmente, além de abranger vários tipos de aplicação, a OWASP disponibiliza vários outros projetos voltados para diversas áreas da segurança da informação.

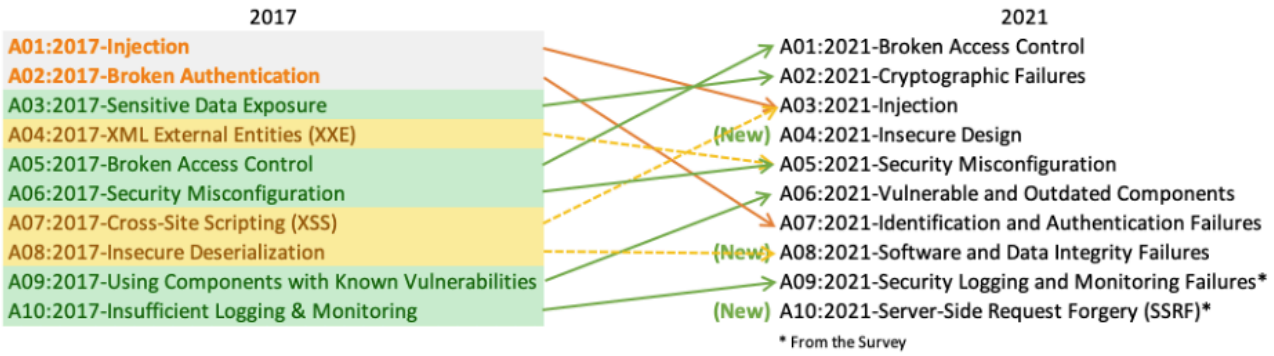
O OWASP opera sob um modelo de 'comunidade aberta', onde qualquer pessoa pode participar e contribuir com projetos, eventos, chats online e muito mais. Um princípio guia da OWASP é que todos os materiais e informações são gratuitos e de fácil acesso no site, para todos. OWASP oferece desde documentação, ferramentas, vídeos, fóruns, projetos, até eventos.

Resumindo, o OWASP é um repositório de todas as coisas relacionadas à segurança de aplicações, apoiado pelo amplo conhecimento e experiência de seus colaboradores da comunidade aberta.

O OWASP Top Ten é um documento online, disponibilizado no site do OWASP, que fornece classificação e orientação de remediação para os 10 principais riscos de segurança de aplicações web. O relatório é baseado em um consenso entre especialistas em segurança de todo o mundo. Os riscos são classificados com base na frequência das falhas de segurança descobertas, na gravidade das vulnerabilidades e na magnitude de seus impactos potenciais. O objetivo do relatório é oferecer aos desenvolvedores e profissionais de segurança de aplicações uma visão dos riscos de segurança mais prevalentes, para que possam incorporar as descobertas e recomendações do relatório em suas práticas de segurança, minimizando assim a presença desses riscos conhecidos em seus aplicativos.

OWASP mantém a lista dos top 10 desde 2003. A cada 2-3 anos, em média, a lista é atualizada de acordo com os avanços e mudanças no mercado de AppSec. A importância do OWASP está na informação acionável que fornece, ela serve como uma lista de verificação chave e padrão interno de desenvolvimento de aplicações para Web e está presente em milhares de organizações pelo mundo.

A versão mais recente foi lançada em 2021 e incluiu mudanças significativas em relação à versão de 2017, conforme mostrado na figura abaixo. Os problemas de injeção continuam sendo um dos problemas de segurança mais vulneráveis no aplicativo, e a exposição de dados confidenciais aumentou em importância. Alguns novos problemas foram adicionados, como desserialização insegura, e alguns outros problemas foram mesclados.



Quadro comparativo da versão atual da OWASP Top Ten com a versão anterior - fonte: https://owasp.org/www-project-top-ten/

- A01:2021 - Broken Access Control
- A02:2021 - Cryptographic Failures
- A03:2021 - Injection (Cross-site Scripting is now part of this category in this edition.)
- A04:2021 - Insecure Design (is a new category for 2021, with a focus on risks related to design flaws)
- A05:2021 - Security Misconfiguration
- A06:2021 - Vulnerable and Outdated Components (was previously titled Using Components with Known Vulnerabilities)
- A07:2021 - Identification and Authentication Failures (was previously Broken Authentication)
- A08:2021 - Software and Data Integrity Failures (is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity) (Insecure Deserialization from 2017 is now a part of this larger category.)
- A09:2021 - Security Logging and Monitoring Failures (was previously Insufficient Logging & Monitoring)
- A10:2021 - Server-Side Request Forgery

O Projeto Application Security Verification Standard (ASVS) fornece uma base para testar os controles técnicos de segurança de aplicações Web e também fornece aos desenvolvedores uma lista de requisitos para desenvolvimento seguro.

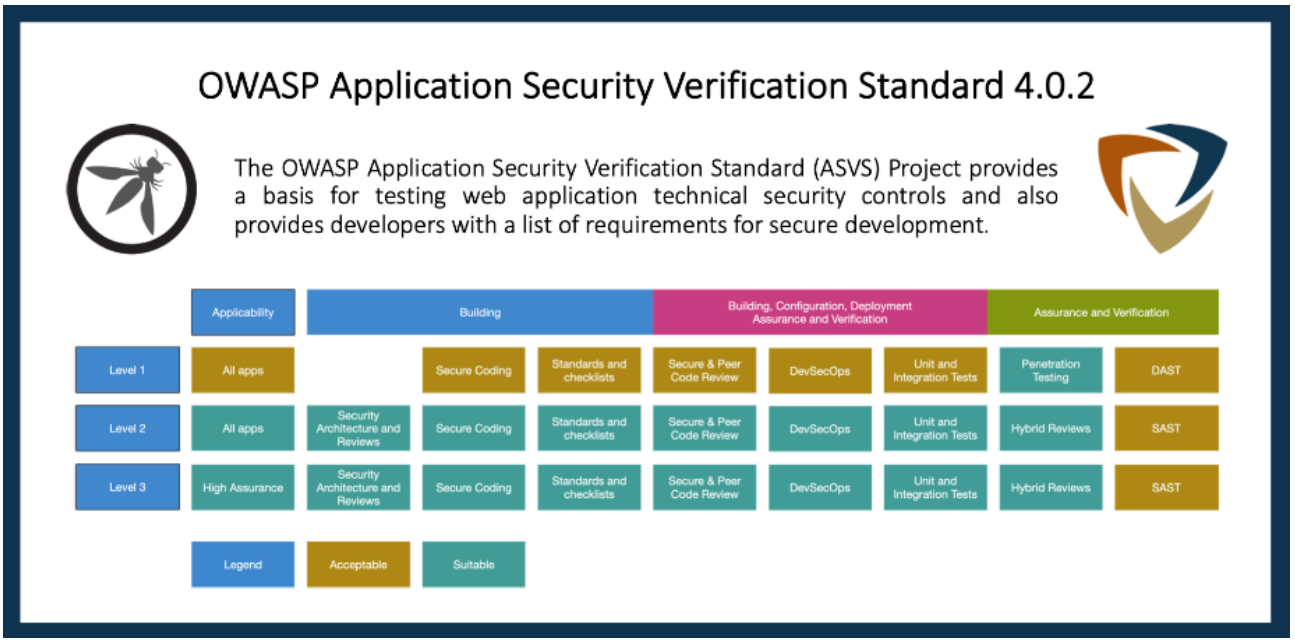
O objetivo principal do ASVS é normalizar o alcance da cobertura e o nível de rigor disponíveis no mercado quando se trata de realizar a verificação de segurança de aplicações Web usando um padrão aberto comercialmente viável. O padrão fornece uma base para testar os controles técnicos de segurança da aplicação, bem como quaisquer controles técnicos de segurança no ambiente, que são usados para proteger contra vulnerabilidades como Cross-Site Scripting (XSS) e injeção de SQL. Esse padrão pode ser usado para estabelecer um nível de confiança na segurança de aplicações Web. Os requisitos foram desenvolvidos com os seguintes objetivos em mente:

- **Use como uma métrica** - Forneça aos desenvolvedores e proprietários das aplicações um parâmetro para avaliar o grau de confiança que pode ser colocado em suas aplicações Web.
- **Use como orientação** - Fornece orientação aos desenvolvedores de controle de segurança sobre o que construir nos controles de segurança para satisfazer os requisitos de segurança da aplicação.
- **Uso durante a aquisição** - Fornece uma base para especificar os requisitos de verificação de segurança do aplicativo em contratos.

O padrão fornecido pelo ASVS funciona como uma diretriz durante todo o ciclo de vida da aplicação, onde ajuda a definir os requisitos de segurança antecipadamente, integrá-los a um SDLC (Software Development Lifecycle) e verificar se os mesmos estão sendo atendidos adequadamente. Também pode ser usado para avaliar e verificar a segurança de uma aplicação já existente.

O ASVS possui três níveis de requisitos que são sugeridos conforme a criticidade do sistema para o negócio:

- **Nível 1:** é o nível mais básico e que toda aplicação não-transacional deveria utilizar, como sites e blogs. Pode ser validado por meio de um pentest.
- **Nível 2:** recomendado para aplicações e sistemas que contêm dados pessoais e é o nível recomendado para a maior parte das aplicações.
- **Nível 3:** recomendado para aplicações críticas e transacionais, que contém dados pessoais sensíveis, como registros médicos, dados financeiros ou qualquer outra aplicação que requeira um alto nível de confiabilidade.



ASVS version 4.0.2 - Fonte: https://owasp.org/www-project-application-security-verification-standard/

Os principais resultados obtidos pelo uso dos controles OWASP ASVS são:

- Estabelecer níveis de controles de segurança com base na criticidade dos sistemas e aplicações.
- Adotar um processo de acreditação de segurança de software.
- Reduzir o custo com a segurança de aplicações e despesas operacionais.
- Estabelecer requisitos de segurança para o desenvolvimento seguro de software por terceiros e fábricas de software.
- Atendimento de normativos, como a Seção 6.5 (desenvolvimento seguro) do PCI-DSS 3.2.1

A documentação do ASVS define vários requisitos e verificações necessárias para cada etapa de segurança da aplicação. As empresas costumam possuir dificuldades em quais itens devem ser priorizados e como acompanhar a evolução da segurança durante o desenvolvimento da aplicação.

OWASP SAMM - Software Assurance Maturity Model, foi criado com a missão de fornecer uma maneira eficaz e mensurável para todos os tipos de organizações analisarem e melhorarem sua postura de segurança de software. Ele visa conscientizar e educar as organizações sobre como projetar, desenvolver e implantar software seguro por meio de modelo próprio de autoavaliação. O SAMM suporta todo o ciclo de vida do software (SDLC) e é independente de tecnologia e processo.

O SAMM foi construído para ser evolutivo e orientado a riscos, pois não existe uma receita única que funcione para todas as organizações. Dessa forma, ele foi pensado para ser:

- MENSURÁVEL - Níveis de maturidade definidos em todas as práticas de segurança
- ACIONÁVEL - Caminhos claros para melhorar os níveis de maturidade
- VERSÁTIL - Independente de tecnologia, processo e organização

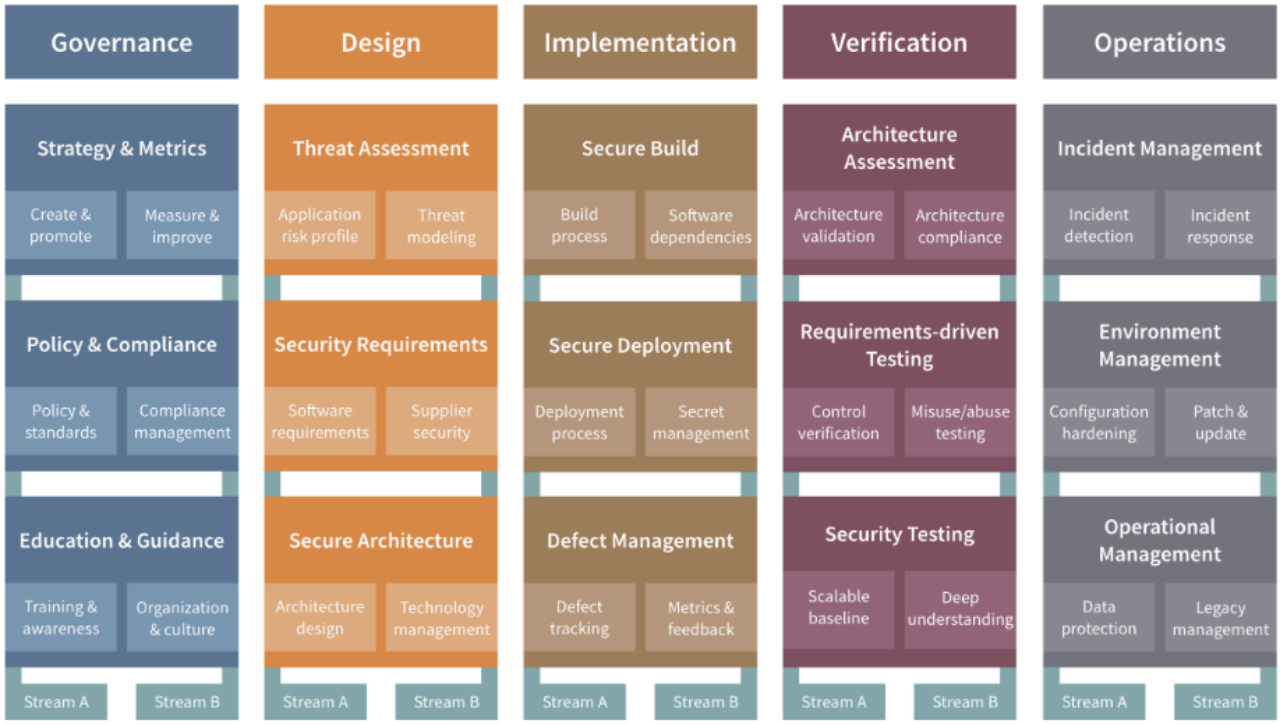
O Modelo OWASP SAMM

O SAMM é um modelo prescritivo, uma estrutura aberta que é simples de usar, totalmente definida e mensurável. Os detalhes da solução são fáceis de seguir, mesmo para o pessoal que não é de segurança. Ele ajuda as organizações a analisar suas práticas atuais de segurança de software, criar um programa de segurança em iterações definidas, mostrar melhorias progressivas nas práticas seguras e definir e medir atividades relacionadas à segurança.

Ele foi definido com flexibilidade em mente para que pequenas, médias e grandes organizações que usam qualquer estilo de desenvolvimento possam personalizá-lo e adotá-lo. Ele fornece um meio de saber onde sua organização está em sua jornada em direção à garantia de software e entender o que é recomendado para avançar para o próximo nível de maturidade.

Ele não insiste que todas as organizações atinjam o nível máximo de maturidade em todas as categorias. Cada organização pode determinar o nível de maturidade desejado para cada prática de segurança que melhor se adequa e adaptar os modelos disponíveis para suas necessidades específicas.

O SAMM é baseado em cerca de 15 práticas de segurança agrupadas em 5 funções de negócios. Cada prática de segurança contém um conjunto de atividades, estruturadas em 3 níveis de maturidade. As atividades em um nível de maturidade mais baixo geralmente são mais fáceis de executar e exigem menos formalização do que as de um nível de maturidade mais alto.



Modelo SAMM - fonte: https://owasp.org/www-project-samm/

O que é Monitoramento em DevOps?

No desenvolvimento de software, o monitoramento em DevOps é a prática de rastrear e medir o desempenho e a integridade de sistemas e aplicações para identificar e corrigir problemas antecipadamente. Isso inclui a coleta de dados sobre tudo, desde a utilização da CPU até o espaço em disco e os tempos de resposta da aplicação. Ao identificar problemas antecipadamente, o monitoramento em DevOps pode ajudar as equipes a evitar interrupções ou degradação do serviço.

De muitas maneiras, isso pode parecer semelhante ao tipo de monitoramento usado em qualquer operação de TI bem projetada. No entanto, o monitoramento em DevOps é mais profundo. A metodologia DevOps orienta as equipes por ciclos curtos de planejamento, desenvolvimento, implantação e revisão/avaliação. O monitoramento em DevOps, para ser totalmente integrado, precisará, portanto, ser um monitoramento contínuo.

Então, o que é monitoramento contínuo? O monitoramento contínuo é o processo de verificação regular e vigilante de sistemas, redes e dados em busca de sinais de degradação de desempenho. Realizado manualmente ou automaticamente, o monitoramento contínuo normalmente envolve o uso de software para verificar vulnerabilidades e rastrear alterações nas configurações de segurança. O monitoramento contínuo visa identificar ameaças potenciais antecipadamente, abordando-as antes que se tornem um problema.

Por que monitorar DevOps?

À medida que uma empresa adota uma cultura e abordagem de DevOps – aumentando a comunicação e a colaboração ao derrubar a barreira entre o desenvolvimento e as operações – o monitoramento é uma prática fundamental para detectar problemas com um sistema antes que eles venham a ocorrer. O monitoramento eficaz ajuda a resolver suas preocupações em relação à eficiência do desenvolvimento e à complexidade do sistema:

- Precisamos enviar o código mais rapidamente, mas como garantir que não introduzimos vulnerabilidades ocultas em nosso código?
- Nosso sistema é construído com tantas partes móveis. Como posso ficar de olho em tudo?
- O projeto inteiro às vezes parece uma caixa preta impenetrável. Como obter a visibilidade de que preciso?

Se você deseja criar melhorias em áreas como balanceamento de carga e segurança, ou se deseja criar ferramentas de processo para coisas como protocolos de reversão e infraestrutura de auto recuperação, precisará de monitoramento para ajudá-lo a ver dentro de seus aplicativos e sua infraestrutura. O monitoramento de DevOps pode fornecer uma solução

clara, fácil de consumir e de painel único, melhorando tanto o processo de entrega de software quanto o software final a ser entregue.

Em termos de seu processo de entrega de software, o monitoramento pode ajudá-lo a determinar linhas de base (e melhorias subsequentes) para indicadores-chave de desempenho (KPIs), como:

- Frequência de implantação
- Falhas de implantação
- Número de erros de código
- Tempo de ciclo de solicitação de pull request
- Alterar taxa de falha
- Tempo médio entre falhas (MTBF)
- Tempo médio para detecção (MTTD) de erros

Com visibilidade por meio do monitoramento, você terá uma visão e controle aprimorados sobre suas operações, e isso o posicionará para entregar aplicativos funcionais e confiáveis no prazo.

Casos de uso

Cada estágio da sua produção de DevOps deve ser visível. Isso inclui uma visão geral da integridade e das atividades da plataforma de infraestrutura. No entanto, mesmo as menores unidades de valor – por exemplo, uma única linha de código – precisam de sua atenção. Nos exemplos a seguir, vamos tentar cobrir as principais funções envolvidas nesse processo.

- **Linting de código -** As ferramentas de linting de código analisam seu código quanto a estilo, sintaxe e possíveis erros. Em muitos casos, eles também verificam as melhores práticas e a conformidade com um padrão de codificação. Linting pode ajudá-lo a encontrar e corrigir problemas em seu código antes que eles causem erros de tempo de execução ou outros problemas. Linting também ajuda a garantir que seu código seja limpo e consistente.
- **Operações de fluxo de trabalho Git -** Os conflitos de base de código podem ocorrer quando dois ou mais desenvolvedores tentam trabalhar na mesma parte de um projeto ao mesmo tempo. O Git tem vários recursos que podem ajudá-lo a gerenciar e resolver conflitos, incluindo commits e rollbacks. Ao monitorar as operações de fluxo de trabalho do git quanto a conflitos, você pode garantir que seu projeto permaneça coeso e consistente.
- **Logs de integração contínua (CI) -** Os logs de CI podem ajudar a determinar se suas compilações de código estão sendo executadas com êxito ou se ocorreram erros ou avisos. Se houver erros, eles exigirão recursos para investigar, solucionar problemas e corrigir. Além disso, o monitoramento de seus logs pode ajudá-lo

a identificar possíveis problemas com seu pipeline de compilação ou base de código que precisam ser resolvidos.

- **Logs de pipeline de implantação contínua (CD) -** O monitoramento de seus logs de CD pode fornecer informações valiosas sobre a integridade e o status do pipeline. Ao monitorar os logs, você pode solucionar problemas de implantações com falha e identificar possíveis problemas.
- **Logs de alterações de gerenciamento de configuração -** Seus logs de alterações de gerenciamento de configuração podem fornecer informações valiosas sobre o estado do sistema e as alterações críticas. Ao monitorar esses logs, você pode rastrear alterações manuais e automatizadas feitas em seus sistemas, identificar alterações não autorizadas e solucionar problemas.
- **Logs de implantação de infraestrutura -** Seus logs de implantação rastreiam quando novas pilhas são implantadas e se elas falharam. Esses logs podem ajudar a solucionar problemas com implantações de pilha e também identificar alterações não autorizadas na infraestrutura que podem ter causado uma falha.
- **Instrumentação de código -** A instrumentação de código é o processo de adicionar código ao seu aplicativo para coletar dados sobre seu desempenho e caminho de operação. Com a instrumentação em vigor, você pode rastrear chamadas de pilha e ver valores contextuais. O monitoramento da saída de instrumentação de código permite medir a eficácia de suas práticas de DevOps e identificar quaisquer áreas que precisem de melhorias. Também pode ajudar a identificar bugs e auxiliar nos testes.
- **Rastreamento distribuído -** O rastreamento distribuído é fundamental para monitorar e depurar aplicativos de microsserviços. Ao entender como seus aplicativos interagem uns com os outros (geralmente por meio de APIs), fica mais fácil identificar e corrigir problemas. O rastreamento distribuído também pode ajudá-lo a otimizar o desempenho de seu aplicativo, identificando gargalos.
- **Monitoramento de desempenho de aplicativos (APM) -** O APM rastreia o desempenho e a disponibilidade dos aplicativos. Isso pode incluir rastreamento de tempos de resposta, monitoramento de erros, Real User Monitoring (RUM) para rastreamento de experiência do usuário final e muito mais. Ao usar plataformas APM, você pode identificar e corrigir problemas antes que eles causem problemas no restante do sistema.
- **Monitoramento de acesso à API -** Ao rastrear e registrar o acesso e o tráfego da API, você pode identificar e impedir o acesso não autorizado ou possíveis ataques DDoS

- **Monitoramento de infraestrutura -** O monitoramento de infraestrutura rastreia o desempenho e a disponibilidade de sistemas e redes de computadores. As ferramentas de monitoramento de infraestrutura podem fornecer informações em tempo real sobre métricas, como utilização de CPU, espaço em disco, memória e tráfego de rede. Essas ferramentas podem ajudar a identificar problemas de recursos antes que causem uma interrupção ou outro problema.
- **Monitoramento de rede -** O monitoramento de rede rastreia o desempenho e a disponibilidade de uma rede de computadores e seus componentes individuais. Os administradores de rede usam ferramentas de monitoramento de rede para identificar problemas com a rede e tomar medidas corretivas. O monitoramento de rede também usa logs de fluxo de rede para identificar quaisquer atividades suspeitas.
- **Monitoramento sintético -** O monitoramento sintético é um tipo de teste de software, usando representações virtuais de sistemas e componentes do mundo real. O monitoramento sintético pode testar o desempenho, a funcionalidade e a confiabilidade de componentes individuais do sistema ou de um sistema inteiro.

Application Performance Management (APM)

O que é?

O gerenciamento de desempenho de aplicações (APM) permite que as organizações prevejam e evitem problemas de desempenho antes que eles afetem seus usuários ou seus negócios.

O APM ajuda uma organização a garantir que suas aplicações críticas atendam às expectativas estabelecidas de desempenho, disponibilidade e experiência do cliente ou do usuário final. Ele faz isso medindo o desempenho da aplicação, alertando os administradores quando as linhas de base de desempenho não são atendidas, fornecendo visibilidade das causas principais dos problemas de desempenho e resolvendo automaticamente muitos problemas de desempenho antes que eles afetem os usuários ou os negócios.

APM também é uma abreviação para monitoramento de desempenho de aplicações. Os termos são frequentemente usados de forma intercambiável, mas o monitoramento de desempenho de aplicativos é, na verdade, um componente de muitos gerenciamentos de desempenho de aplicativos - porque, afinal, você precisa monitorar o desempenho para gerenciá-lo.

Cada vez mais, no entanto, as soluções de gerenciamento de desempenho de aplicações estão evoluindo a depender de ferramentas tradicionais de monitoramento de desempenho de aplicações, para incorporar observabilidade, uma tecnologia de coleta e análise de dados de desempenho mais adequada aos cenários mais modernos com aplicações nativas de nuvem com maior complexidade.

Dimensões de Serviços de APM

As dimensões de serviços de APM podem ser divididas em:

- **End user experience -** O monitoramento da experiência do usuário final pode ser realizada de duas maneiras: sintética/proativa ou real. A mais comum é a sintética/proativa, que é feita via robôs emuladores, que simulam o comportamento do usuário na forma real, por meio de logs de aplicação ou plugins de softwares especializados em APM. Nos dois tipos a monitoração é utilizada para gerar dashboards e disparar alarmes, caso ocorra alguma violação de tempo ou SLA (Service Level Agreement).
- **Runtime application architecture -** O uso do runtime application architecture para conhecer o fluxo das transações é de extrema importância, pois cada vez mais as aplicações se encontram

distribuídas e descentralizadas. Ter um desenho transacional atualizado de forma automática, faz parte da entrega de algumas das muitas ferramentas de APM do mercado.

- **Business transactions -** Como exemplo de business transactions, podemos citar: quantidade de pedidos, quantidade de vendas de um e-commerce por hora, quantidade de chamadas de uma API. Saber as principais business transactions do seu negócio e seus requisitos de desempenho, é papel do profissional que lida diariamente com ferramentas de APM. Em momentos de crise de desempenho e disponibilidade, saber os acordos de níveis de serviço (SLAs) definidos para cada uma das business transactions é essencial para ter conhecimento sobre o status de um serviço ou componente de TI.
- **Deep Dive Component Monitoring -** Ao monitorar todos componentes de software e hardware, de forma detalhada, a resolução de problemas de desempenho se torna mais rápida e efetiva. A informação sobre todos esses componentes, pode auxiliar no tratamento de causa raiz dos problemas. Entretanto, é importante ter cuidado para não gerar sobrecarga nas aplicações, pois muitas vezes esses dados são gerados com a instrumentalização das aplicações.
- **Analytics/Reporting -** Normalmente, após a coleta de dados diretamente nas aplicações e infraestruturas que as sustentam, as soluções de APM fornecem ferramentas para analytics e reporting. Nesse caso, dados brutos são coletados para uma posterior análise acerca do desempenho, capacidade ou custos (caso em cloud) de uma aplicação.

Monitoramento de desempenho de aplicação

No monitoramento do desempenho da aplicação, os agentes são implantados em todo o ambiente da aplicação e na infraestrutura de suporte, para 'monitorar' o desempenho por amostragem de desempenho e métricas relacionadas ao desempenho (às vezes chamadas de telemetria), geralmente com uma frequência de uma vez a cada minuto. Esses agentes realizam:

- **O monitoramento da experiência digital –** reúne métricas de desempenho - como tempo de carregamento, tempo de resposta, tempo de atividade, tempo de inatividade - da interface do usuário no dispositivo do usuário final. (Isso costumava ser chamado de monitoramento da experiência do usuário final, mas foi ampliado para reconhecer que entidades não humanas, como robôs ou outros componentes de software, também interagem com o aplicativo e têm suas próprias expectativas de desempenho). O

monitoramento de experiência digital geralmente suporta monitoramento de usuário real, que monitora a experiência de um usuário real no sistema, e monitoramento sintético, para testes de desempenho em ambientes de produção e não produção.

- **O monitoramento de aplicações –** inclui o monitoramento de toda a pilha de aplicativos - estrutura de aplicativos (por exemplo, Java ou .NET), sistema operacional, banco de dados, APIs, middleware, servidor de aplicativos da Web, UI - bem como monitoramento de infraestrutura de TI que mostra fatores como utilização de CPU, espaço em disco e desempenho da rede. O monitoramento de pilha geralmente inclui rastreamento em nível de código, o que pode ajudar a identificar partes do código que podem estar causando um gargalo de desempenho.
- **O monitoramento de banco de dados** – mostra o desempenho de consultas ou procedimentos SQL, além do monitoramento de banco de dados fornecido pelos agentes de monitoramento de aplicativos.
- **O monitoramento de disponibilidade** – monitora a disponibilidade real de componentes de aplicativos e hardware (porque os aplicativos podem gerar dados de desempenho mesmo quando não estão acessíveis ao usuário final).

Além de coletar dados de desempenho, esses agentes realizam perfis de transação definidos pelo usuário, rastreando cada transação da interface do usuário ou dispositivo do usuário final por meio de cada componente ou recurso da aplicação envolvida na transação. Essas informações são usadas para determinar as dependências da aplicação e para criar um mapa de topologia - uma visualização das dependências entre os componentes da aplicação e da infraestrutura local, na nuvem e ou ambientes híbridos.

As soluções de APM normalmente fornecem um controlador e um painel centralizado onde as métricas de desempenho coletadas são agregadas, analisadas e comparadas com as linhas de base estabelecidas. O painel alerta os administradores do sistema sobre desvios das linhas de base que indicam problemas de desempenho reais ou potenciais; ele também fornece informações contextuais e insights acionáveis que os administradores podem usar para solucionar e solucionar os problemas.

Observability

O que é?

Em geral, observabilidade é a medida em que você pode entender o estado ou condição interna de um sistema complexo com base apenas no conhecimento de suas saídas externas. Quanto mais observável um sistema, mais rápida e precisa você pode navegar de um problema de desempenho identificado para sua causa raiz, sem testes ou codificação adicionais.

Na computação em nuvem, observabilidade também se refere a ferramentas e práticas de software para agregar, correlacionar e analisar um fluxo constante de dados de desempenho de uma aplicação distribuída e do hardware em que ela é executada, a fim de monitorar, solucionar problemas e depurar com mais eficiência a aplicação para atender melhor às expectativas de experiência dos clientes, acordos de nível de serviço (SLAs) e outros requisitos de negócios.

Por ser um tópico relativamente novo, a observabilidade é muitas vezes descaracterizada como uma palavra de ordem exagerada ou uma 'rebranding' do monitoramento do sistema em geral e do monitoramento de desempenho de aplicativos (APM) em particular. Na verdade, a observabilidade é uma evolução natural dos métodos de coleta de dados do APM que abordam melhor a natureza cada vez mais rápida, distribuída e dinâmica das implantações de aplicações nativas da nuvem. A observabilidade não substitui o monitoramento, ela permite um melhor monitoramento e um melhor APM.

O termo 'observabilidade' vem da teoria do controle, uma área da engenharia preocupada em automatizar o controle de um sistema dinâmico - por exemplo, o fluxo de água através de um cano ou a velocidade de um automóvel em inclinações e descidas - com base no feedback do sistema.

Por que precisamos de observabilidade?

Nos últimos anos, as equipes de TI confiaram principalmente no APM para monitorar e solucionar problemas em aplicações. O APM periodicamente mostra e agrega dados de aplicações e sistemas, chamados de telemetria, que são conhecidos por estarem relacionados a problemas de desempenho nas aplicações. Ele analisa a telemetria em relação aos principais indicadores de desempenho (KPIs) e reúne os resultados em um painel para alertar as operações e as equipes de suporte sobre condições anormais que devem ser abordadas para resolver ou evitar problemas.

O APM é eficaz o suficiente para monitorar e solucionar problemas de aplicações monolíticas ou aplicações distribuídas, onde novos códigos são

lançados periodicamente e fluxos de trabalho e dependências entre componentes de aplicações, servidores e recursos relacionados são bem conhecidos e fáceis de rastrear.

Mas hoje as organizações estão adotando rapidamente práticas modernas de desenvolvimento (desenvolvimento ágil, integração contínua e entrega contínua (CI/CD), DevOps, várias linguagens de programação) e tecnologias nativas da nuvem, como microsserviços, contêineres e funções sem servidor (serverless). Como resultado, eles estão trazendo mais serviços ao mercado mais rápido do que nunca. Mas no processo eles estão implantando novos componentes de aplicações com tanta frequência, em tantos lugares, em tantas linguagens diferentes e por períodos de tempo tão variados (por segundos ou frações de segundo, no caso de funções sem servidor) que nos APMs, a amostragem de dados não consegue acompanhar o ritmo.

O que é necessário é uma telemetria de alta qualidade que possa ser usada para criar um registro totalmente correlacionado, rico em contexto e de alta fidelidade de cada solicitação ou transação do usuário da aplicação. Para isso, utilizamos a observabilidade.

Como funciona a observabilidade?

As plataformas de observabilidade descobrem e coletam a telemetria de desempenho continuamente, integrando-se à instrumentação existente incorporada aos componentes de aplicativos e infraestrutura e fornecendo ferramentas para adicionar instrumentação a esses componentes. A observabilidade se concentra em quatro tipos principais de telemetria:

- **Logs –** Os logs são registros granulares, com carimbo de data/hora, completos e imutáveis de eventos do aplicativo. Entre outras coisas, os logs podem ser usados para criar um registro milissegundo a milissegundo de alta fidelidade de cada evento, completo com o contexto circundante, que os desenvolvedores podem 'reproduzir' para fins de solução de problemas e depuração.
- **Metrics –** As Métricas (às vezes chamadas de métricas de série temporal) são medidas fundamentais da integridade do aplicativo e do sistema durante um determinado período de tempo, como quanta memória ou capacidade de CPU um aplicativo usa em um período de cinco minutos ou quanta latência um aplicativo experimenta durante um período de tempo.
- **Traces –** Os rastreamentos registram a 'jornada' de ponta a ponta de cada solicitação do usuário, da interface do usuário ou aplicativo móvel por toda a arquitetura distribuída e de volta ao usuário.

- **Dependências –** As dependências (também chamadas de mapas de dependência) revelam como cada componente do aplicativo depende de outros componentes, aplicativos e recursos de TI.

Depois de coletar essa telemetria, a plataforma a correlaciona em tempo real para fornecer às equipes de DevOps, equipes de engenharia de confiabilidade do site (SREs) e equipes de TI, informações contextuais completas – o quê, onde e por quê de qualquer evento que possa indicar, causar ou ser usado para resolver um problema de desempenho do aplicativo.

Muitas plataformas de observabilidade descobrem automaticamente novas fontes de telemetria que podem surgir no sistema (como uma nova chamada de API para outro software). E porque eles lidam com muito mais dados do que uma solução APM padrão, muitas plataformas incluem recursos de AIOps (inteligência artificial para operações) que separam os sinais (indicações de problemas reais) de ruído (dados não relacionados a problemas).

Benefícios da observabilidade

O benefício abrangente da observabilidade é que, um sistema mais observável é mais fácil de entender, mais fácil de monitorar, mais fácil e seguro de atualizar com novo código é mais fácil de reparar do que um sistema menos observável. Mais especificamente, a observabilidade suporta diretamente os objetivos Agile/DevOps de fornecer software de alta qualidade mais rapidamente, permitindo que uma organização:

- **Descubra e resolva 'unknown unknowns' (riscos/problemas que você não sabe que existem) –** Uma limitação principal das ferramentas de monitoramento é que elas apenas observam 'known unknowns' (riscos/problemas que você já sabe que existem), condições excepcionais que você já sabe observar. A observabilidade descobre condições que você talvez nunca conheça ou pense em procurar e, em seguida, rastreia sua relação com problemas de desempenho específicos e fornece o contexto para identificar as causas-raiz para acelerar a resolução.
- **Capture e resolva problemas no início do desenvolvimento –** A observabilidade inclui o monitoramento nas fases iniciais do processo de desenvolvimento de software. As equipes de DevOps podem identificar e corrigir problemas no novo código antes que eles afetem a experiência do cliente ou os SLAs.
- **Escale a observabilidade automaticamente –** Por exemplo, você pode especificar a instrumentação e a agregação de dados como parte de uma configuração de cluster do Kubernetes e começar a

coletar a telemetria desde o momento em que ela gira até que ela desce.

- **Habilite a correção automatizada e a infraestrutura de aplicativos de auto recuperação –** Combine a observabilidade com os recursos de aprendizado de máquina e automação de AIOps (Artificial Intelligence for IT Operations) para prever problemas com base nas saídas do sistema e resolvê-los sem intervenção do gerenciamento.

Artigos

- Observability (IBM Learn platform)
- APM (IBM Learn platform)
- DevOps Monitoring
- Observability vs. APM vs. Monitoring.