DEVOPS E INTEGRAÇÃO CONTÍNUA

AULA 5

Prof. Mauricio Antonio Ferste

CONVERSA INICIAL

Nesta etapa sobre Princípios de Segurança em DevOps, mergulharemos nos fundamentos essenciais para integrar efetivamente a segurança em todo o ciclo de vida do desenvolvimento e operações. Exploraremos a importância da Integração Contínua de Segurança, destacando como a automação de testes e a criação de uma cultura organizacional centrada na segurança são cruciais para o desenvolvimento de sistemas robustos.

Ao abordar temas como monitoramento em tempo real, gestão de identidade e acesso, conformidade e resposta a incidentes, ofereceremos insights práticos para construir uma abordagem holística que protege os sistemas contra ameaças e promove a resiliência operacional.

Prepare-se para uma jornada que redefine a maneira como concebemos a segurança em ambientes DevOps, integrando-a como um elemento-chave para o sucesso sustentável.

TEMA 1 – PRINCÍPIOS DE SEGURANÇA EM DEVOPS

Segurança em DevOps aprofunda-se na abordagem integrada para garantir a segurança ao longo de todo o ciclo de vida do desenvolvimento e operações de software. Ele explora os fundamentos da Integração Contínua de Segurança, delineando como a automação de testes de segurança, análises estáticas e dinâmicas, e revisões de código são essenciais para a detecção precoce e a mitigação de vulnerabilidades.

Além disso, esta etapa aborda estratégias para promover uma cultura de segurança, envolvendo colaboradores em práticas seguras e incentivando a responsabilidade compartilhada. Ao discutir temas como monitoramento, gestão de identidade e resposta a incidentes, oferece uma visão abrangente para integrar com sucesso a segurança no contexto dinâmico do DevOps, garantindo robustez e confiança nas operações de TI modernas.

- A conscientização sobre segurança é promovida entre todas as equipes de desenvolvimento e operações. A educação contínua sobre práticas seguras é fundamental para manter uma cultura de segurança.

1.1 Integração e entrega contínuas

Relembrando, a entrega contínua representa um método automatizado e confiável de disponibilizar software no ambiente produtivo, em contraste com o modelo tradicional que espera até que o software esteja totalmente pronto. Na abordagem convencional, a interação com o produto só ocorre no final do desenvolvimento, muitas vezes resultando em demoras para implementar alterações necessárias. Em contrapartida, a entrega contínua fragmenta o software em componentes menores, lançados regularmente no ambiente produtivo. Esse enfoque integra a experiência do usuário ao processo de desenvolvimento, facilitando alterações ágeis e seguras. As práticas de entrega contínua minimizam o tempo e os riscos relacionados à implementação de novas versões do software, promovendo a colaboração efetiva entre desenvolvedores, testadores e equipe operacional responsável pela entrega.

De maneira ampla, segundo Fowler (2013), a entrega contínua representa um conjunto de procedimentos e princípios que devem ser seguidos para viabilizar a publicação de um projeto de desenvolvimento de software no ambiente produtivo a qualquer momento.

Essa abordagem difere do processo tradicional de desenvolvimento de software, que implica entregar o software completo apenas quando totalmente pronto.

No modelo convencional, a interação com o produto só ocorre no final do desenvolvimento, potencialmente gerando a necessidade de alterações com demora significativa para conclusão. Por outro lado, na entrega contínua, o software é fracionado em partes menores, sendo regularmente publicado no ambiente produtivo. Isso assegura que a experiência do usuário seja incorporada ao desenvolvimento, possibilitando alterações de maneira mais ágil e segura.

- Importante! Esse é o momento crítico de nossa entrega, que é rigorosamente conduzida segundo o modelo ágil. No modelo ágil, o conceito de “pronto” é fundamental e define claramente que, para ser considerado completo, cada bloco entregue deve estar devidamente testado e alinhado com padrões de qualidade estritos.

1.2 Implementação segura e ciclo de vida seguro

Práticas seguras são aplicadas na implantação de código, incluindo a configuração adequada de ambientes e a adoção de boas práticas de segurança para proteger a infraestrutura em nuvem. Essas medidas são incorporadas ao ciclo de vida de desenvolvimento, desde a definição de requisitos até o desenvolvimento, teste e implantação. Isso garante uma abordagem holística à segurança.

1.3 Monitoramento contínuo

A implementação de monitoramento contínuo é crucial para identificar atividades suspeitas ou anomalias de segurança em tempo real. Isso inclui a análise de logs, métricas de segurança e a adoção de soluções de monitoramento avançadas.

1.4 Gestão de Identidade e Acesso (IAM)

A gestão rigorosa de identidade e acesso é crucial. Princípios de menor privilégio, autenticação multifatorial e monitoramento de acessos são implementados para evitar acessos não autorizados.

1.5 Resposta rápida a incidentes

Protocolos de resposta a incidentes são estabelecidos, garantindo uma ação rápida e eficiente em caso de violações de segurança. Isso inclui a comunicação transparente e a implementação de correções imediatas.

1.6 Testes automatizados

Com ciclos de entrega mais curtos, a realização manual de todos os testes necessários torna-se praticamente inviável, destacando a importância dos testes automatizados no contexto do DevOps.

Os testes automatizados constituem uma atividade multifuncional que envolve toda a equipe de desenvolvimento, sendo executados de forma contínua desde o início do projeto. Seu propósito é assegurar que as funcionalidades sejam implementadas de maneira correta e abrangente, cobrindo uma ampla gama de comportamentos do software.

Dentro do ecossistema DevOps, os responsáveis pelos testes colaboram estreitamente com desenvolvedores e usuários para mapear, redigir e automatizar testes desde as fases iniciais do projeto. Essa abordagem possibilita que os testes sejam delineados em conjunto com os requisitos, antes mesmo de os desenvolvedores iniciarem qualquer atividade.

Os benefícios dos testes automatizados no contexto do DevOps são numerosos, incluindo a redução do tempo de entrega, uma vez que esses testes podem ser executados de maneira mais ágil do que os testes manuais. Além disso, contribuem para diminuir o risco de falhas ao identificar possíveis problemas no software antes de sua publicação no ambiente produtivo, resultando em uma melhoria significativa da qualidade do software desenvolvido.

A entrega contínua representa um método automatizado e confiável de disponibilizar software no ambiente produtivo, em contraste com o modelo tradicional que espera até que o software esteja totalmente pronto.

Na abordagem convencional, a interação com o produto só ocorre no final do desenvolvimento, muitas vezes resultando em demoras para implementar alterações (Humble; Farley, 2010).

Segundo HPE (2016), quando os requisitos de usuário e os testes que os validam não estão vinculados, as mudanças nos requisitos podem levar a testes falhos que não mostrarão quantos requisitos foram atendidos. Um dos princípios de DevOps é investir em automação para permitir a execução de tarefas de forma mais rápida, para minimizar a possibilidade de falha humana e se tornar um processo mais confiável.

Com o aumento da cultura DevOps e o crescimento da colaboração entre equipes de operações e desenvolvedores, diversas ferramentas surgiram e têm evoluído para garantir a padronização e automação da gestão de infraestrutura. Essas ferramentas permitem que a infraestrutura seja administrada da mesma maneira que os desenvolvedores de softwares tratam o código fonte do programa, pelo controle de versões, realização de testes automatizados, distribuição de módulos comuns e execução de mudanças de configuração nos servidores (Sato, 2014).

1.7 Testes automatizados com Terraform

O Terraform usa uma sintaxe de arquivos de configuração chamada HCL (que veremos mais adiante) para definir os recursos de infraestrutura que deseja

criar. Esses arquivos de configuração são tratados como código-fonte do projeto, com controle de versão e controle do código-fonte incluídos.

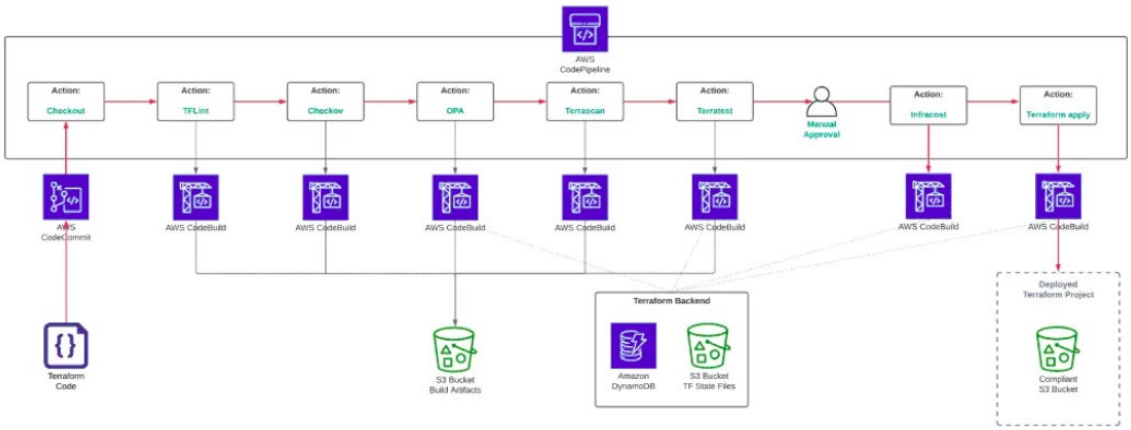
Para implantar uma infraestrutura definida em um arquivo de configuração do Terraform, você pode usar o comando terraform apply. O Terraform irá gerar um plano de implantação que você pode visualizar antes de aplicar. Os testes são uma parte importante do processo de gerenciamento de infraestrutura em nuvem. Os testes podem ajudar a garantir que a infraestrutura esteja funcionando corretamente e que esteja em conformidade com os requisitos.

O Terraform oferece suporte a diversos tipos de testes, desempenhando um papel crucial na garantia de que a infraestrutura funcione de maneira adequada e esteja em conformidade com os requisitos estabelecidos.

Para relembrar:

- **Testes de unidade**: visam examinar partes individuais da infraestrutura e geralmente são escritos pelo desenvolvedor responsável pela área específica em teste. Exemplos incluem verificar se uma função de infraestrutura retorna o valor correto ou se lança uma exceção ao receber um argumento inválido;
- **Testes de integração**: voltados para avaliar a interação entre partes distintas da infraestrutura, os testes de integração são geralmente elaborados por equipes de desenvolvedores colaborando em um projeto de infraestrutura. Exemplos incluem verificar se duas funções de infraestrutura podem se comunicar efetivamente ou se uma alteração em uma função não afeta outras;
- **Testes de aceitação**: têm como objetivo avaliar a infraestrutura como um todo e são frequentemente elaborados por usuários ou clientes que utilizarão a infraestrutura. Exemplos incluem verificar se a infraestrutura atende aos requisitos do usuário e se é fácil de usar;
- **Testes de integração contínua**: na integração contínua (CI), os testes de integração são executados automaticamente sempre que há uma modificação na base de código, assegurando que as alterações no código não afetem adversamente a infraestrutura. Exemplos incluem a execução de ferramentas de análise estática de código, o comando “terraform validate” para verificar a sintaxe do arquivo de configuração e o comando “terraform plan” para confirmar se a configuração opera conforme o esperado.

Figura 1 – Como é o processo de qualidade utilizando o Terraform



Fonte: Maksimov, 2021.

A ilustração fornecida apresenta um diagrama que descreve o ciclo de vida de um recurso gerenciado pelo Terraform, dividido em cinco etapas principais.

Na primeira etapa, denominada Planejamento, o Terraform analisa a configuração do recurso e gera um plano de execução detalhando as modificações necessárias na infraestrutura, permitindo revisão para garantir correspondência com as expectativas.

Em seguida, na etapa de Aplicação, o plano é executado para provisionar ou atualizar o recurso na infraestrutura, utilizando APIs dos provedores de nuvem.

A etapa de Monitoramento vem em seguida, onde o Terraform é empregado para monitorar o estado do recurso, verificando seu funcionamento e conformidade com a configuração desejada.

Quando mudanças são necessárias, o Terraform gera um novo plano refletindo as alterações desejadas, que pode ser aplicado na etapa de Mudanças.

Por fim, na etapa de Destruição, o Terraform remove o recurso da infraestrutura e libera os recursos associados quando não é mais necessário.

Esse diagrama oferece uma visão do processo completo de gerenciamento de recursos pelo Terraform, proporcionando uma abordagem consistente e independente do provedor de nuvem. Os testes desempenham uma função vital no processo de gerenciamento de infraestrutura com o Terraform, contribuindo para garantir que a infraestrutura opere corretamente e esteja em conformidade com os requisitos estabelecidos.

- Por exemplo, veja a seguir testes usando Terraform.
    - <https://learn.microsoft.com/pt-br/azure/developer/terraform/best-practices-compliance-testing>.

TEMA 2 – FERRAMENTAS PARA ANÁLISE E GESTÃO DE RISCOS

Na dinâmica e complexa paisagem tecnológica atual, em que a infraestrutura digital desempenha um papel crucial em operações empresariais, a análise e gestão de riscos tornam-se imperativas. Ferramentas especializadas emergem como aliadas essenciais, permitindo às organizações a identificação, avaliação e mitigação de potenciais ameaças à segurança e à estabilidade operacional.

Dentro desse contexto, o Terraform, uma ferramenta de Infraestrutura como Código (IaC), destaca-se como uma peça fundamental. Além de sua capacidade intrínseca de orquestrar ambientes de nuvem, o Terraform contribui significativamente para a gestão proativa de riscos. Esse artigo explorará a integração do Terraform como uma ferramenta de análise e gestão de riscos, destacando como sua abordagem orientada a código impacta positivamente na segurança e confiabilidade da infraestrutura digital.

2.1 Análise teórica sobre riscos

A análise e gestão de riscos é uma parte importante do gerenciamento de qualquer infraestrutura, incluindo infraestrutura em nuvem. O Terraform pode ser usado para ajudar a automatizar e simplificar o processo de análise e gestão de riscos.

2.2 Análise de riscos

A análise de riscos é o processo de identificação, avaliação e mitigação de riscos. O Terraform pode ser usado para ajudar na análise de riscos de infraestrutura de várias maneiras, incluindo:

- **Identificação de riscos**: o Terraform pode ser usado para gerar uma lista de todos os recursos de infraestrutura que estão sendo gerenciados. Essa lista pode ser usada como ponto de partida para a identificação de riscos;
- **Avaliação de riscos**: o Terraform pode ser usado para avaliar os riscos de cada recurso de infraestrutura. Isso pode ser feito usando uma

variedade de métodos, como análise de vulnerabilidade, análise de impacto nos negócios e análise de custo-benefício;

- **Mitigação de riscos**: o Terraform pode ser usado para implementar medidas para mitigar os riscos identificados. Isso pode incluir coisas como o uso de controles de segurança, a implementação de backups e o uso de ferramentas de monitoramento.

2.3 Gestão de riscos

A gestão de riscos é o processo de monitoramento e gerenciamento dos riscos identificados. O Terraform pode ser usado para ajudar na gestão de riscos de infraestrutura de várias maneiras, incluindo:

- **Monitoramento de riscos**: o Terraform pode ser usado para monitorar os riscos identificados. Isso pode incluir coisas como o uso de ferramentas de monitoramento, a execução de auditorias e a realização de exercícios de resposta a incidentes;
- **Gerenciamento de mudanças**: o Terraform pode ser usado para gerenciar mudanças na infraestrutura. Isso pode ajudar a garantir que as mudanças não aumentem os riscos;
- **Resposta a incidentes**: o Terraform pode ser usado para responder a incidentes de segurança. Isso pode incluir coisas como o uso de ferramentas de resposta a incidentes, a implementação de planos de recuperação de desastres e a comunicação com as partes interessadas.

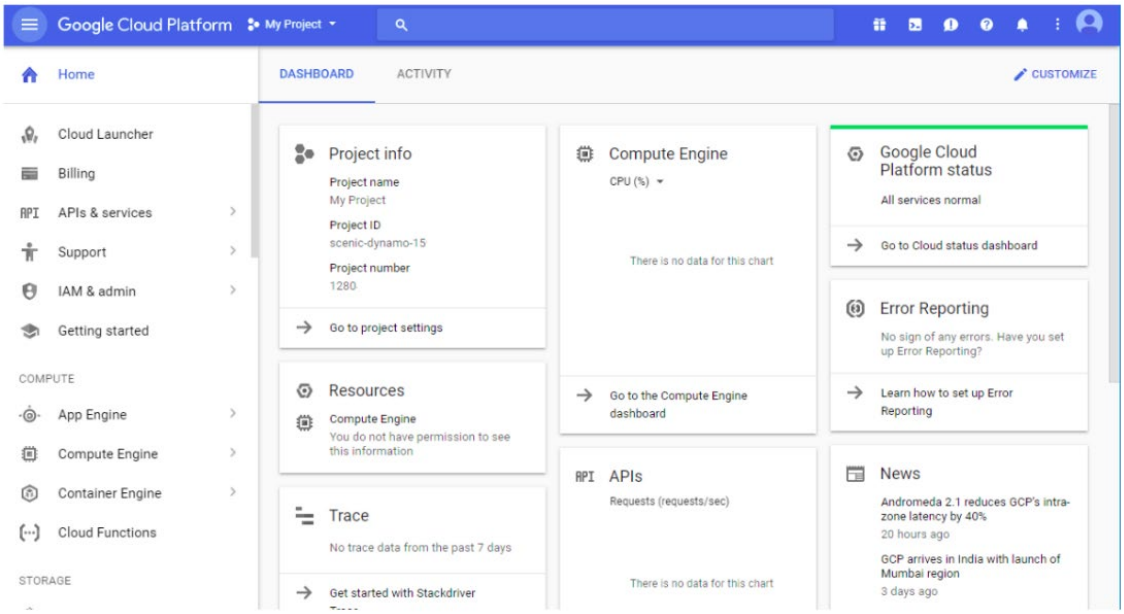
Exemplos de riscos que podem ser analisados com o Terraform:

- **Riscos de segurança**: vulnerabilidades de segurança, acesso não autorizado, comprometimento de dados;
- **Riscos de disponibilidade**: interrupções de serviço, falhas de hardware, falhas de software;
- **Riscos de desempenho**: baixo desempenho, sobrecarga de recursos, latência alta;
- **Riscos de custo**: custos excessivos, recursos subutilizados, recursos ociosos.

Vantagens da análise e gestão de riscos com o Terraform, veja na Figura 2 um exemplo de tela.

- **Automatização**: o Terraform pode automatizar muitas das tarefas envolvidas na análise e gestão de riscos, o que pode economizar tempo e recursos;
- **Precisão**: o Terraform pode ajudar a garantir que a análise e gestão de riscos sejam precisas e completas;
- **Visibilidade**: o Terraform pode fornecer uma visão abrangente dos riscos de infraestrutura, o que pode facilitar a tomada de decisões.

Figura 2 – Dashboard para análise com várias plataformas



Fonte: Klein, 2022.

Considerações para a análise e gestão de riscos com o Terraform:

- **Competência**: é importante ter a competência necessária para usar o Terraform para analisar e gerenciar riscos;
- **Escalabilidade**: o Terraform pode ser usado para gerenciar infraestruturas de qualquer tamanho, mas é importante considerar a escalabilidade ao implementar a análise e gestão de riscos;
- **Custo**: o Terraform é uma ferramenta gratuita, mas é importante considerar os custos associados à implementação da análise e gestão de riscos.

No geral, o Terraform pode ser uma ferramenta valiosa para ajudar a automatizar e simplificar o processo de análise e gestão de riscos de infraestrutura.

2.4 Análise prática com exemplo sobre riscos

Imagine que uma equipe está utilizando o Terraform para provisionar a infraestrutura na AWS para uma aplicação web crítica. Vamos realizar uma análise de riscos focada em alguns aspectos específicos do uso dessas tecnologias.

- **Exposição de credenciais**:
  - **Risco**: uso indevido ou exposição acidental de credenciais da AWS;
  - **Mitigação**: implementação rigorosa de práticas de segurança IAM, evitando o armazenamento de chaves no código Terraform. Utilização de variáveis seguras e armazenamento seguro do arquivo de estado.
- **Configurações de segurança incorretas**:
  - **Risco**: configurações de segurança inadequadas em recursos AWS, como grupos de segurança mal configurados;
  - **Mitigação**: revisão frequente das configurações no código Terraform, execução de análises de segurança automáticas e adoção de políticas de segurança específicas.
- **Inconsistências no ambiente**:
  - **Risco**: divergência entre a configuração planejada no Terraform e a configuração real na AWS;
  - **Mitigação**: utilização de **terraform plan** antes da aplicação para validar as alterações planejadas. Implementação de monitoramento contínuo para identificar desvios não autorizados.
- **Exposição de dados sensíveis**:
  - **Risco**: inclusão acidental de dados sensíveis no código Terraform;
  - **Mitigação**: utilização de variáveis sensíveis, criptografia adequada e revisão cuidadosa do código antes da aplicação. Armazenamento seguro de variáveis sensíveis usando mecanismos como AWS Secrets Manager.
- **Falhas nas atualizações de segurança**:
  - **Risco**: falha em aplicar patches e atualizações de segurança apropriados;

    - **Mitigação**: automação de processos de atualização utilizando serviços como AWS Systems Manager Patch Manager. Implementação de práticas de DevSecOps para garantir que atualizações de segurança sejam parte integrante do ciclo de vida da infraestrutura.
- **Dependências externas vulneráveis**:
    - **Risco**: utilização de módulos Terraform ou recursos de terceiros com vulnerabilidades de segurança;
    - **Mitigação**: verificação regular de vulnerabilidades em módulos utilizados, acompanhamento de atualizações e implementação de práticas de controle de versão.
- **Controle de acesso inadequado**:
    - **Risco**: concessão excessiva de permissões a usuários ou recursos;
    - **Mitigação**: revisão regular das políticas IAM, princípios do menor privilégio e implementação de auditorias regulares.

Realizar uma análise de riscos periódica, considerando os aspectos específicos do ambiente AWS provisionado pelo Terraform, é fundamental para garantir a segurança contínua da infraestrutura. A abordagem proativa na identificação e mitigação de riscos contribui para a construção de ambientes seguros e resilientes na nuvem.

TEMA 3 – CONFIABILIDADE E CONTINUIDADE EM DEVOPS

A busca por confiabilidade e continuidade em ambientes DevOps é crucial para garantir estabilidade e entrega consistente de software. No contexto da confiabilidade, a atenção recai sobre monitoramento proativo, automação de testes, resiliência e feedback rápido, visando identificar e corrigir potenciais problemas antes que impactem os usuários.

A continuidade em DevOps, por sua vez, concentra-se na integração e implantação contínuas, rollbacks eficientes, backup e recuperação, além da orquestração de contêineres. A integração contínua assegura a validação contínua do código, enquanto a implantação contínua automatiza a entrega de novas funcionalidades.

Estratégias robustas de backup e recuperação são essenciais, proporcionando rápida restauração em casos de desastres. A eficácia dessas

práticas é enfatizada na busca por uma operação ininterrupta, permitindo adaptação rápida a mudanças e promovendo uma experiência de desenvolvimento consistente e confiável.

Em resumo, a confiabilidade e a continuidade em DevOps são alcançadas por meio de automação, resiliência e práticas eficientes, promovendo a entrega contínua de valor ao usuário.

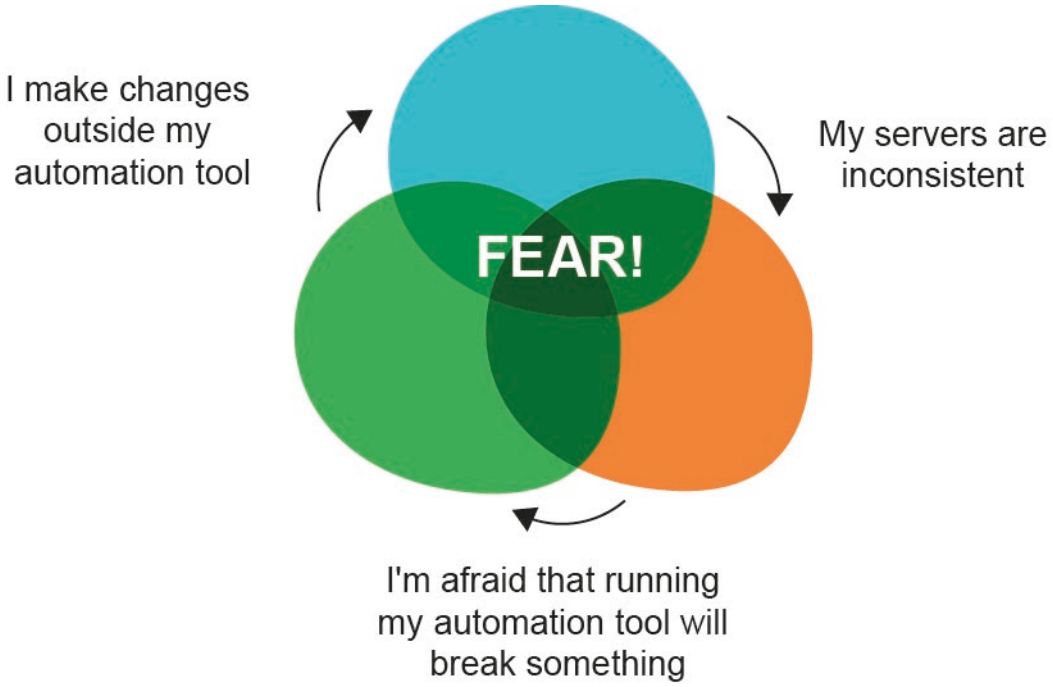
3.1 Contornando Problemas

Conforme mencionado por Morris (2016), a Infraestrutura como Código (IaC) enfrenta desafios, destacando-se o rápido aumento de recursos gerenciados por uma equipe. Esse crescimento acelerado pode resultar na dificuldade de manter todos os servidores atualizados de maneira uniforme, levando a desvios de configuração. Tais desvios ocorrem quando pequenas modificações são implementadas em um sistema específico após a criação uniforme de todos os sistemas.

Os desvios de configuração têm o potencial de causar impactos significativos durante as próximas atualizações, uma vez que o sistema modificado individualmente pode não ser reconhecido. Esses servidores, conhecidos como "flocos de neve", não têm suas alterações refletidas no código que provisiona a infraestrutura, resultando na impossibilidade de replicar sua criação de maneira idêntica em casos de falhas.

De acordo com Morris (2016), a presença de diversos servidores “flocos de neve” em uma infraestrutura a torna frágil. Além disso, as alterações manuais não apenas contribuem para a formação desses servidores singulares, mas também podem gerar receios ou resistência ao uso de ferramentas de automação.

Figura 3 – O ciclo do medo



Fonte: Habbema, 2023.

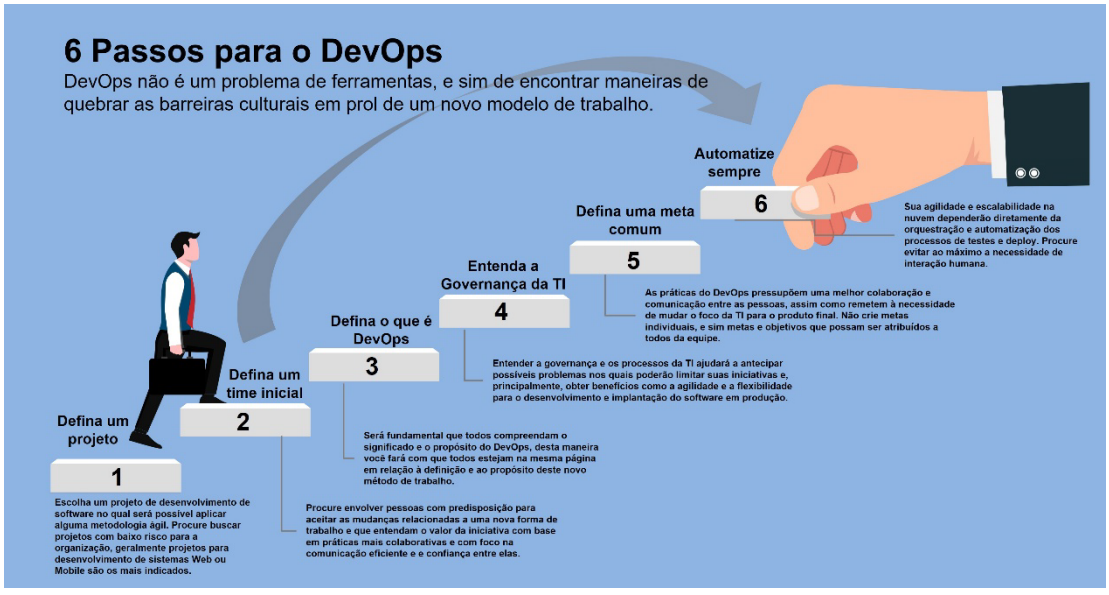
Morris (2016) afirma que a melhor forma de quebrar a espiral de medos é enfrentando-os, deve-se escolher alguns servidores, definir suas configurações de forma que se saiba que estejam funcionais e promover testes a fim de certificar que a configuração possa ser aplicada de forma confiável e possíveis problemas possam ser detectados rapidamente.

Em termos de continuidade, temos de ter um padrão, segue uma versão de como iniciar e manter o uso de DevOps, como na Figura 4, na qual temos um guia prático para Implementação do DevOps em seis etapas:

- **Definição do projeto ideal**: inicie o processo de implementação do DevOps escolhendo um projeto de software de baixo risco, como um sistema web ou mobile. Dê preferência a projetos que se alinhem com metodologias ágeis, garantindo uma transição mais suave para a equipe;
- **Constituição do time adequado**: monte uma equipe composta por indivíduos abertos a mudanças, valorizando colaboração, comunicação eficaz e confiança. Busque uma formação multidisciplinar, incluindo experiência em gestão de projetos ágeis, desenvolvimento com ênfase em integração contínua e testes, além de operação com conhecimento em automação e integração;

- **Adoção da cultura DevOps**: fomente um entendimento claro e unificado sobre os princípios do DevOps e seus benefícios. Estimule a equipe a se tornar propagadora dessa nova cultura de trabalho, disseminando-a por todos os setores de TI e na organização como um todo;
- **Compreensão da governança da TI**: analise minuciosamente os processos e controles de TI para identificar potenciais obstáculos à agilidade e flexibilidade do DevOps. Familiarize-se com procedimentos como controle de mudanças, configuração, atualização, incidentes e problemas, comunicação, auditoria e gerenciamento de projetos. Obtenha o respaldo da alta gestão de TI para introduzir mudanças que facilitem a adoção do DevOps;
- **Estabelecimento de metas compartilhadas**: defina metas e objetivos que abranjam toda a equipe, promovendo a colaboração e foco no produto final. Mensure o progresso e compare o novo modelo de trabalho com o tradicional na área de TI;
- **Automatização como prioridade**: coloque a automatização de testes e implantações como prioridade para garantir escalabilidade na nuvem e eliminar a necessidade de interações humanas repetitivas. Utilize ferramentas de integração contínuas para otimizar o processo de desenvolvimento de forma eficiente.

Figura 4 – Uma visão de seis passos para o DevOps



Crédito: Diki/Adobe Stock.

TEMA 4 – AUTENTICAÇÃO E AUTORIZAÇÃO

No cenário dinâmico e ágil da implementação DevOps, em que a colaboração eficiente e a entrega contínuas são fundamentais, a gestão eficaz de acessos e permissões se torna uma prioridade crítica. Autenticação e autorização emergem como pilares essenciais para garantir a segurança e o controle adequado sobre os recursos de tecnologia da informação. A autenticação refere-se à verificação da identidade dos usuários e sistemas que interagem com a infraestrutura, enquanto a autorização se concentra na concessão adequada de permissões e privilégios para executar ações específicas.

Nesta introdução, exploraremos a importância vital da autenticação e autorização no contexto DevOps, destacando como esses elementos fundamentais contribuem para a construção de um ambiente seguro, eficiente e colaborativo.

4.1 DevSecOps

DevSecOps é uma abordagem que integra a segurança no processo de DevOps. Isso significa que a segurança é considerada em todas as etapas do desenvolvimento e implantação de software, desde o planejamento até a operação.

No DevOps tradicional, a segurança geralmente é considerada uma etapa separada do processo, usualmente no final. Isso pode levar a atrasos e custos adicionais, pois as vulnerabilidades de segurança são descobertas apenas quando o software está quase pronto para ser implantado.

O DevSecOps aborda esse problema integrando a segurança ao processo de DevOps. Isso significa que as avaliações de segurança são realizadas em todas as etapas do processo, desde a definição dos requisitos até a implantação. Assim, a segurança passa a ser uma responsabilidade compartilhada entre todos os membros da equipe envolvidos no desenvolvimento do software. A equipe de desenvolvimento trabalha com a equipe de segurança desde o início do processo, para garantir que os requisitos de segurança sejam considerados. Da mesma forma, após implantar o software, as equipes de operações continuam monitorando-o em busca de problemas de segurança.

O DevSecOps oferece várias vantagens, incluindo:

- **Redução do risco de ataques**: ao integrar a segurança no processo de DevOps, é possível identificar e corrigir vulnerabilidades de segurança mais cedo, o que reduz o risco de ataques;
- **Melhoria da eficiência**: o DevSecOps pode ajudar a melhorar a eficiência, pois permite que as equipes identifiquem e corrijam problemas de segurança mais rapidamente;
- **Garantia da conformidade**: o DevSecOps pode ajudar a garantir a conformidade com os requisitos de segurança, como os padrões de segurança da informação.

DevSecOps representa uma abordagem que incorpora a segurança desde as fases iniciais do desenvolvimento até a implementação de aplicativos. Isso implica que a segurança é uma consideração integral, não apenas uma reflexão tardia no processo. Implementar o DevSecOps requer uma mudança de mentalidade organizacional, em que a segurança é reconhecida como parte essencial do ciclo de desenvolvimento de aplicativos. Além disso, é crucial adotar novas ferramentas e processos para automatizar as práticas de segurança.

As equipes de DevOps devem priorizar a automação da segurança para proteger não apenas o ambiente, mas também os dados e o processo de integração contínua/entrega contínua (CI/CD). Isso engloba medidas de segurança específicas para aplicativos menores executados em containers.

Ao adotar o DevSecOps, as organizações podem identificar e abordar precocemente vulnerabilidades de segurança, reduzindo significativamente o risco de possíveis ataques. Essa abordagem proativa cria uma base sólida para o desenvolvimento e implementação seguros de aplicativos.

Que tal usar ChatGPT? Cada arquivo individual de Infraestrutura como Código (IaC) é lido e transformado em uma string, posteriormente utilizado para parametrizar a pergunta predefinida para o bot ChatGPT na forma da pergunta “Encontre falhas de segurança no script {file_type}: {contents}”. Aqui, o primeiro parâmetro representa um espaço reservado para o tipo de arquivo relacionado ao IaC, enquanto outro contém o conteúdo do próprio script. Uma vez que a pergunta é construída, a solicitação ao ChatGPT é enviada por meio da API Python. Após isso, as respostas retornadas para cada um dos arquivos IaC são resumidas em uma tabela HTML, que é retornada como a saída final de nossa solução. As entradas dessa tabela representam arquivos distintos do arquivo

compactado, e para cada um deles, é fornecido um resumo sobre possíveis falhas geradas pelo ChatGPT. Por outro lado, o ChatGPT também fornece conselhos sobre como resolver os problemas detectados, o que também está incluído como parte do resumo dos resultados.

Um exemplo de detecção de problema e conselhos oferecidos é mostrado na Figura 5.

Figura 5 – Exemplo de exploração de práticas de segurança em Códigos de infraestrutura como código (IaC): análise de variáveis sensíveis e sugestões de melhorias pelo ChatGPT

Fonte: Ferste, 2024.

4.2 Práticas para DevSecOps

- **Integrar a segurança desde o início**: a segurança deve ser incorporada desde os estágios iniciais do desenvolvimento de software. Isso implica que as equipes de segurança colaborem com as equipes de desenvolvimento para identificar e mitigar riscos desde o início do processo;
- **Manter a segurança após a implementação**: a importância da segurança não termina com a implementação do software. As equipes de operações desempenham um papel vital, monitorando o software em busca de vulnerabilidades e ataques após a implantação;
- **Adotar ferramentas de segurança automatizadas**: ferramentas de segurança automatizadas desempenham um papel crucial em agilizar a verificação de segurança. Esse aspecto é particularmente relevante para equipes de DevOps que enfrentam a necessidade de realizar múltiplas revisões em um mesmo dia;

- **Cultivar consciência de segurança**: é essencial que toda a equipe de desenvolvimento de software esteja consciente da importância da segurança. Isso implica que cada membro da equipe esteja familiarizado com os riscos de segurança e compreenda como mitigá-los.

O Amazon Inspector, um serviço oferecido pela Amazon Web Services (AWS), desempenha um papel crucial na segurança de ambientes hospedados na AWS, proporcionando uma avaliação automatizada de possíveis vulnerabilidades em aplicações e recursos. Lançado com o objetivo de oferecer segurança avançada, o Inspector executa análises contínuas em instâncias do Amazon EC2, aplicações web e arquiteturas baseadas em microservices.

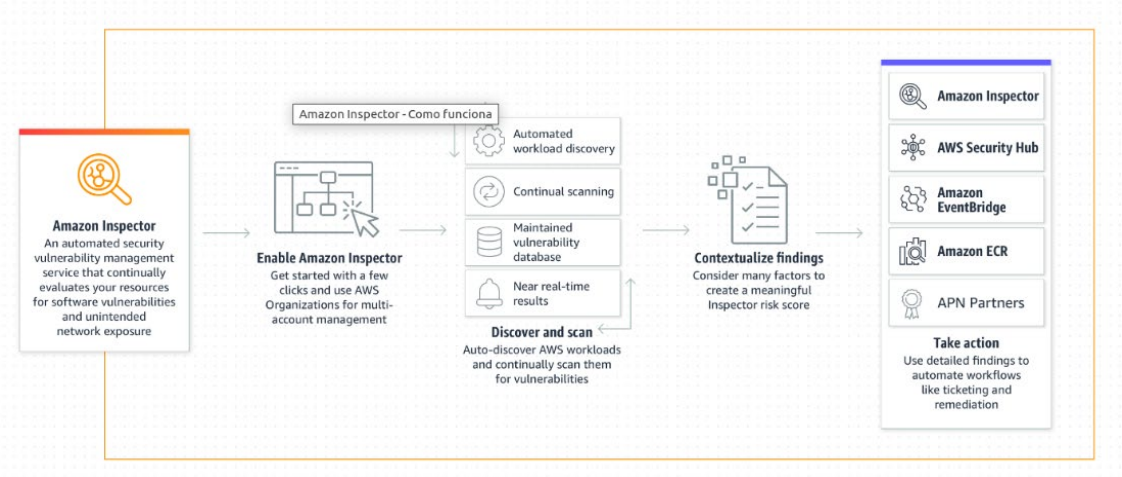
Uma das características-chave do Amazon Inspector é a capacidade de realizar avaliações contínuas de segurança, garantindo que potenciais ameaças e vulnerabilidades sejam identificadas à medida que o ambiente evolui. Além disso, o serviço realiza varreduras automáticas de segurança, examinando a configuração do sistema operacional, as aplicações instaladas e a rede para identificar brechas potenciais.

O Amazon Inspector baseia suas avaliações em normas de segurança predefinidas ou personalizadas, permitindo a verificação da conformidade do ambiente em relação às melhores práticas de segurança. Os relatórios detalhados gerados pelo serviço fornecem informações sobre possíveis ameaças, com recomendações específicas para corrigir as vulnerabilidades identificadas.

Outro destaque é a capacidade de integração com outros serviços da AWS, como o AWS CloudWatch, para monitoramento e registro contínuos das atividades de segurança. Além disso, o Amazon Inspector oferece suporte a diversos sistemas operacionais, incluindo Linux e Windows, tornando-o uma solução versátil para avaliações de segurança em diferentes ambientes.

Em resumo, o Amazon Inspector é uma ferramenta essencial para organizações que buscam fortalecer a segurança de suas aplicações e recursos na AWS. Ao automatizar a identificação de vulnerabilidades e fornecer insights detalhados, o serviço desempenha um papel significativo na implementação de práticas de segurança proativas e na mitigação de riscos em ambientes de nuvem.

Figura 6 – Exemplo de segurança com AWS inspector abaixo e seu esquema de construção



Fonte: Amazon Inspector, 2023.

Para utilizar o Amazon Inspector com o Terraform, você pode seguir os passos seguintes para implementar a integração:

- Certifique-se de ter o Terraform instalado em sua máquina e configure suas credenciais da AWS no seu ambiente local, utilizando o AWS CLI ou configurando diretamente o arquivo ~/.aws/credentials.

  Criação de um Perfil do Amazon Inspector com o Terraform:
- Utilize o Terraform para criar um perfil do Amazon Inspector. Um exemplo de código Terraform pode ser semelhante a este código HCL:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_inspector_assessment_target" "example" {
  name = "example-target"
}

resource "aws_inspector_assessment_template" "example" {
  name               = "example-template"
  duration           = 3600
  rules_package_arns = ["arn:aws:inspector:us-west-
2:758058086616:rulespackage/0-9hgA516p"]
}
```

Esse exemplo cria um alvo de avaliação e um modelo de avaliação. Certifique-se de substituir os valores pelos apropriados para o seu caso de uso.

Execução do Terraform:

- Execute o comando **terraform init** e, em seguida, **terraform apply** para aplicar as alterações. Isso criará o perfil do Amazon Inspector.

  Configuração do AWS Inspector:
- Após a execução do Terraform, acesse o console do AWS Inspector e configure suas avaliações conforme necessário.

  Execução de avaliações:
- Use o Terraform para automatizar a execução de avaliações. Exemplo:

```
resource "aws_inspector_run" "example" {
  assessment_template_arn =
aws_inspector_assessment_template.example.arn
  assessment_target_arn   =
aws_inspector_assessment_target.example.arn
}
```

Isso criará e executará uma avaliação utilizando o modelo e o alvo definidos anteriormente. Lembre-se de ajustar esses exemplos de acordo com suas necessidades específicas e seguir as práticas recomendadas de segurança, como o gerenciamento seguro de credenciais e a configuração adequada de políticas de acesso na AWS.

TEMA 5 – SEGURANÇA COMO CÓDIGO

Essa abordagem de “Segurança como Código” redefine a maneira como as organizações encaram a proteção de seus sistemas. Incorporando a segurança diretamente no código e nos processos de desenvolvimento, essa metodologia preconiza a integração de práticas de segurança desde o início até a operação contínua. Isso se traduz na automação de verificações de segurança, utilizando ferramentas especializadas nos pipelines de CI/CD, além da extensão da prática à infraestrutura por meio do conceito de Infraestrutura como Código (IaC).

Ao tratar políticas de segurança como elementos codificados e promover a conscientização contínua dentro das equipes, a abordagem de “Segurança como Código” busca fortalecer a cultura de segurança e proporcionar respostas ágeis a ameaças potenciais.

Ao adotar a mentalidade de “Segurança como Código”, as organizações não apenas abordam as vulnerabilidades desde a concepção, mas também promovem uma abordagem proativa na identificação e correção de falhas de

segurança. Essa estratégia, incorporada ao código-fonte e ao fluxo de desenvolvimento, reforça a segurança de forma contínua, proporcionando uma resposta ágil e eficaz às ameaças emergentes.

A integração de práticas de segurança no código, a automação de verificações e a conscientização contínua são pilares fundamentais dessa abordagem, visando criar ambientes de software mais resilientes e seguros ao longo de todo o ciclo de vida do desenvolvimento.

5.1 Genericamente

As equipes de software usam as ferramentas de DevSecOps a seguir para avaliar, detectar e relatar falhas de segurança durante o desenvolvimento de software.

- **Teste de segurança de aplicações estáticas**
    - As ferramentas de teste de segurança de aplicações estáticas (SAST) analisam e encontram vulnerabilidades no código-fonte proprietário.
- **Análise de composição de software**
    - A análise de composição de software (SCA) é o processo de automatizar a visibilidade do uso de software de código aberto (OSS) para fins de gerenciamento de riscos, segurança e conformidade de licenças.
- **Teste de segurança de aplicações interativas**
    - As equipes de DevSecOps usam ferramentas de teste de segurança de aplicações interativas (IAST) para avaliar as possíveis vulnerabilidades de uma aplicação no ambiente de produção. O IAST consiste em monitores de segurança especiais que são executados com base na aplicação.
- **Teste de segurança de aplicações dinâmicas**
    - As ferramentas de teste de segurança de aplicações dinâmicas (DAST) imitam hackers ao testar a segurança da aplicação de fora da rede.

5.2 Na prática com AWS e Terraform

A segurança em código tornou-se uma prioridade essencial em ambientes de nuvem, especialmente ao utilizar serviços da Amazon Web Services (AWS) em conjunto com ferramentas como o Terraform. Essa abordagem, também conhecida como Infrastructure as Code (IaC), não apenas permite a automação eficiente da infraestrutura, mas também impulsiona a implementação de práticas seguras desde a concepção até a operação.

Vamos explorar como a segurança pode ser integrada efetivamente ao código ao utilizar AWS e Terraform.

1. Gerenciamento de credenciais

O primeiro passo crítico é assegurar o gerenciamento seguro de credenciais na AWS. Utilizar os serviços IAM (Identity and Access Management) é fundamental, criando roles e políticas que concedem acesso mínimo necessário. O Terraform pode ser configurado para utilizar essas credenciais de maneira segura, evitando a exposição indevida de chaves de acesso.

2. Práticas de segurança em Terraform

No âmbito do Terraform, práticas específicas garantem a segurança em código. Isso inclui:

- **Variáveis sensíveis e arquivos de estado**: evitar a exposição de informações sensíveis, como chaves de acesso, utilizando variáveis e armazenando o arquivo de estado do Terraform de forma segura, por exemplo, em um backend remoto com criptografia;
- **Avaliação de impacto**: antes de aplicar alterações, utilizar ferramentas como **terraform plan** para avaliar o impacto potencial e garantir que as modificações não introduzam vulnerabilidades inadvertidas;
- **Revisão de código**: implementar revisões de código para garantir que as configurações do Terraform estejam alinhadas com as políticas de segurança da organização.

3. Monitoramento e conformidade

Implementar monitoramento contínuo para identificar e reagir a possíveis violações de segurança. Serviços como AWS Config podem ser integrados para avaliação contínua da conformidade com políticas de segurança.

4. Auditorias e revisões regulares

Realizar auditorias regulares nas configurações do Terraform e nos recursos provisionados na AWS para identificar e corrigir potenciais lacunas de

segurança. Essas revisões devem incluir análises de políticas de segurança, controle de acesso e configurações específicas de serviços AWS.

5. Atualizações e Patches automáticos

Automatizar a aplicação de patches e atualizações de segurança para os recursos provisionados. Isso inclui a utilização de recursos como o AWS Systems Manager e AWS Systems Manager Patch Manager para garantir que os sistemas estejam sempre atualizados com as correções mais recentes.

FINALIZANDO

Ao incorporar as práticas discutidas nesta etapa, as equipes podem criar um ambiente DevSecOps robusto, em que a entrega contínua e a segurança são aliadas inseparáveis, capacitando as organizações a enfrentarem os desafios modernos de maneira eficaz e inovadora. Por fim, destacamos a importância de educar as equipes de desenvolvimento e operações sobre as melhores práticas de segurança em todos os estágios do processo DevSecOps. Uma cultura de segurança sólida é essencial para garantir a conformidade, proteger os ativos da organização e construir sistemas resilientes na nuvem.

A integração segura de credenciais e a gestão de infraestrutura como código:

AWS com Terraform desempenha um papel crítico na implementação eficaz de práticas DevSecOps, em que a segurança é incorporada desde o início do ciclo de vida do desenvolvimento até a entrega contínua. Ao longo desta etapa, exploramos as melhores práticas para gerenciamento de credenciais na AWS e como o Terraform pode ser utilizado de maneira segura para provisionar e gerenciar recursos em nuvem.

A integração de DevSecOps com Terraform e AWS Credenciais não apenas fortalece a segurança, mas também proporciona maior automação, colaboração e rastreabilidade. Ferramentas como AWS Secrets Manager e AWS Systems Manager Parameter Store oferecem soluções robustas para o gerenciamento seguro de segredos, alinhando-se com os princípios fundamentais de segurança no DevSecOps.

Entendemos a importância de seguir as recomendações de segurança ao lidar com credenciais na AWS, tais como o uso de roles IAM e práticas de gerenciamento seguro de chaves de acesso. Essas abordagens ajudam a

reduzir o risco de exposição acidental de credenciais e contribuem para um ambiente mais seguro.

No contexto do Terraform, destacamos a necessidade de manter a separação adequada entre código e configuração sensível. Utilizando variáveis, arquivos de ambiente e mecanismos de gestão de segredos, conseguimos minimizar a exposição de informações sensíveis durante o desenvolvimento, implantação e manutenção de infraestrutura como código.

REFERÊNCIAS

AMAZON CODECOMMIT. **Site**. 2023. Disponível em: <https://aws.amazon.com/pt/codecommit/>. Acesso em: 20 dez. 2023.

AMAZON DEVSECOPS. Site. 2023. Disponível em: <https://aws.amazon.com/pt/what-is/devsecops/>. Acesso em: 29 jan. 2024.

AMAZON INSPECTOR. Site. 2023. Disponível em: <https://aws.amazon.com/inspector/>. Acesso em: 15 nov. 2023.

AMAZON WEB SERVICES. **O que é DevOps?** Disponível em: <https://aws.amazon.com/pt/devops/what-is-devops/>. Acesso em: 5 nov. 2023.

AZURE REPOS. **Site**. 2023. Disponível em: <https://azure.microsoft.com/pt-br/products/devops/repos>. Acesso em: 20 dez. 2023.

BOOCH, G. **Software Engineering with Ada**. Estados Unidos: [*s. n.*], 1986.

BORGES, E. N. **Conceitos e Benefícios do Test Driven Development**. Universidade Federal do Rio Grande do Sul. Disponível em: <http://www.inf.ufrgs.br/~cesantin/TDD-Eduardo.pdf>. Acesso em: 20 jun. 2021.

CAELUM. **Test-Driven Development**. Disponível em: <http://tdd.caelum.com.br/>. Acesso em: 7 dez. 2023.

CARMENA R. M. **How to teach Git**. Disponível em: <https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html>. Acesso em: 19 dez. 2023.

CHACON, S.; STRAUB, B. **Pro Git**. 2. ed. Apress, 2014.

CUKIER, D. **DDD – Introdução a Domain Driven Design**. 2010. Disponível em: <http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/>. Acesso em: 7 dez. 2023.

DHADUK, H. How Netflix Became A Master of DevOps? An Exclusive Case Study. 202. Disponível em: <https://www.simform.com/blog/netflix-devops-case-study/>. Acesso em: 10 nov. 2023.

DEVMEDIA. **TDD**: fundamentos do desenvolvimento orientado a testes. Disponível em: <http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-atestes/28151>. Acesso em: 10 nov. 2023.

_____. **Test Driven Development**: TDD Simples e Prático. 2019. Disponível em: <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 20 dez. 2023.

ELEMARJR. **BDD na prática – parte 1** – Conceitos básicos e algum código. 2012. Disponível em: <https://elemarjr.wordpress.com/2012/04/11/bdd-na-prtica-parte-1-conceitos-bsicos-e-algum-cdigo/>. Acesso em: 10 nov. 2023.

ENAP. **Curso**: O papel do DevOps na Transformação Digital dos Serviços Públicos. 2020.

FOWLER, M. **Site**. Disponível em: <https://martinfowler.com/>. Acesso em: 19 dez. 2023.

FREEMAN, S. PRYCE, N. **Desenvolvimento de Software Orientado a objetos, Guiado por Testes**. Rio de Janeiro: Alta Books, 2012.

FREECODECAMP. **Site**. Disponível em: <https://www.freecodecamp.org/>. Acesso em: 20 dez. 2023.

GASPARETO, O. **Test Driven Development**. Universidade Federal do Rio Grande do Sul. Disponível em: <http://www.inf.ufrgs.br/~cesantin/TDD-Otavio.pdf>. Acesso em: 8 dez. 2023.

GIT-FAST-VERSION-CONTROL. **Site**. 2023. Disponível em: <https://git-scm.com/>. Acesso em: 20 dez. 2023.

GOMES, A. F. **Desenvolvimento Ágil com Kanban**. Disponível em: <http://www.devmedia.com.br/websys.5/webreader.aspat=6&artigo=2955&revista=javamagazine_84#a-2955>. Acesso em: 10 nov. 2023.

Maksimov, A. **Test Terraform code using CICD** – Easy AWS automation. Hands-on. Cloud, 2021. Disponível em: <https://hands-on.cloud/test-terraform-code-examples/>. Acesso em:

HORNBEEK, M. **Engeneering DevOps**, 2019. ISBN: 978-0-1234-6578-8.

HUMBLE, J.; FARLEY, D. **Continuous delivery**: reliable software releases through build, test, and deployment automation. UK: Pearson Education, jul. 2010. (Addison-Wesley Signature Series)

HUTTERMANN, M. **DevOps for Developers**: integrate development and operations. The Agile Way. NY: Apress, 2012.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **The DevOps Handbook**: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. [*S. l.*: *s. n.*], 2016.

KLEIN, A. L. **Aplicação de técnicas de Devops e ferramentas de IAC para gerenciamento de virtualização de computadores**. 79f. Monografia. Disponível em: https://www.univates.br/bduserver/api/core/bitstreams/7b1a8099-26b2-4215-b78a-5b667a2a3ca9/content>. Acesso em: 24 abr. 2024.

NEWMAN. **Building Microservices**. O'Reilly, 2015.

PETOVIĆ, N. **ChatGPT-Based Design-Time DevSecOps**. 2023.

PRESSMAN, R. S.; MAXIM, B. **Software Engineering**: a practitioner's approach. 8. ed. McGraw-Hill, 2014.

ROSA, D. **6 passos para iniciar no DevOps**, 2018. Disponível em: <https://jornadaparanuvem.com.br/6-passos-para-iniciar-na-jornada-do-devops/>. Acesso em: 10 maio 2023.

W3C. **What is MVC?**. Disponível em: <https://www.w3schools.in/mvc-architecture>. Acesso em: 10 maio 2023.

WESLEY P. **Sistemas de controle de versão - SVN e Git**. 2017.

WOLFF, E. **Microservices Flexible Software Architectures**. Leanpub, 2016.