DevOps

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

DevSecOps - Integrando Segurança ao Pipeline de CI/CD

Objetivos da Aula

- Entender o que é **DevSecOps**.
- Aprender a identificar os 3 principais tipos de vulnerabilidades:
  1. **SCA (Análise de Composição de Software):** Vulnerabilidades nas suas dependências (package.json).
  2. **SAST (Teste de Segurança de Aplicação Estática):** Vulnerabilidades no seu próprio código (app.js).
  3. **Verificação de Contêiner:** Vulnerabilidades na sua imagem Docker (ex: no node:22 ou alpine).
- Integrar ferramentas de segurança automaticamente no ci.yml do GitHub Actions.
- Fazer o pipeline **falhar** se uma vulnerabilidade crítica for encontrada.

Ferramentas que Vamos Usar

- **GitHub Actions** (nosso pipeline existente).
- npm audit (para **SCA**, já vem com o Node.js).
- **GitHub Code Scanning** (CodeQL) (para SAST, nativo do GitHub).
- **Trivy** (uma Action popular para verificação de contêineres).

Fase 1: Conceitos e Configuração Inicial (SAST)

Antes de mexer no pipeline, vamos ativar as ferramentas de segurança nativas do GitHub.

1. **O que é SAST?**
    - É uma ferramenta que lê o seu código-fonte (como app.js) e procura por padrões perigosos (como SQL Injection, caminhos de arquivos inseguros, etc.).
2. **Como ativar (SAST):**
    - No seu repositório, clique na aba **Settings**. No menu lateral esquerdo, na seção "**Security**", clique em **Advanced Security**.
    - Na tela que abrir, procure pela seção "**Code scanning**". Você verá a opção "**CodeQL analysis**". Clique no botão "**Set up**".
    - Selecione a opção "**Default**" (Padrão) para que o GitHub configure o pipeline de segurança automaticamente.
    - O GitHub agora vai *automaticamente* escanear seu código em todo push e pull request, de forma separada do seu pipeline de CI/CD.

Fase 1: Conceitos e Configuração Inicial (SAST)

3. Como ativar (Secret Protection e Push protection):

- Role a página até o final, na seção "**Secret Protection**".
- Verifique o botão à direita:
  - Se estiver escrito "**Enable**": Clique para ativar.
  - Se estiver escrito "**Disable": Ótimo**! O recurso já está ativo automaticamente.
- Faça o mesmo logo abaixo em "**Push protection**" (isso impede que você faça git push de uma senha por engano).

Fase 2: Prática - Adicionando Segurança ao Pipeline (SCA e Contêiner)

- Agora, vamos editar o nosso ci.yml. O objetivo é adicionar verificações de segurança no **Job 1 (test-and-build)**.
- **Onde?** As verificações de segurança devem rodar **DEPOIS** dos testes (npm test), mas **ANTES** de construir e publicar a imagem (docker build).
- **Por quê?** Não queremos *nunca* construir ou publicar um artefato (imagem Docker) que sabemos estar inseguro.

Fase 2: Prática - Adicionando Segurança ao Pipeline (SCA e Contêiner)

Vamos usar o npm audit para verificar se alguma de nossas dependências (como express, jest, etc.) tem uma vulnerabilidade conhecida.

1. Abra seu arquivo .github/workflows/ci.yml.
2. No jobs > test-and-build > steps, adicione o seguinte step *depois* do "Rodar Testes":

```
# ... (step 'Rodar Testes' anterior) ...
# 4.5. Executa a Verificação de Dependências (SCA)
- name: Verificar vulnerabilidades de dependências (SCA)
run: npm audit --audit-level=high
# 5. Configura o Docker Buildx (próximo step) ...
```

O que isso faz? npm audit verifica o package-lock.json. O comando --audit-level=high faz com que o pipeline falhe (pare) apenas se encontrar vulnerabilidades "Altas" ou "Críticas".

Fase 2: Prática - Adicionando Segurança ao Pipeline (SCA e Contêiner)

- **Adicionando Verificação de Contêiner**
  - Vamos usar o **Trivy** para escanear a imagem Docker *depois* que ela for construída, mas *antes* de publicá-la no GHCR.
  - Vamos modificar o step "Build e Push da Imagem Docker". Vamos dividi-lo em dois: um step para **Build** e outro para **Push**.
  - Vamos inserir o step do **Trivy entre** eles.
  - No seu ci.yml, substitua pelo código disponível em: https://github.com/raffael-carvalho/ci-cd-sec/blob/main/.github/workflows/ci.yml

Fase 3: Testando o Novo Fluxo DevSecOps

Agora, quando você fizer o fluxo da aula anterior (criar uma feature, abrir PR, fazer o merge para a main):

1. **No Pull Request (CI):**
    - O **CodeQL (SAST)** rodará automaticamente, procurando falhas no seu código.
    - O **test-and-build (Job 1)** rodará:
        - npm test será executado.
        - npm audit (SCA) será executado. Se encontrar uma falha crítica em um pacote, o PR será bloqueado.
        - O pipeline **não** tentará publicar a imagem (correto).

Fase 3: Testando o Novo Fluxo DevSecOps

2. No Merge para a main (CD):

- O **CodeQL (SAST)** rodará novamente na main.
- O **test-and-build (Job 1)** rodará:
  - npm test e npm audit passarão.
  - O **Trivy (Container Scan)** será executado. Se ele encontrar uma vulnerabilidade CRITICAL na sua imagem node:22, o pipeline **falhará**. O push para o GHCR não acontecerá.
  - O **Job 2 (deploy-to-local) não será executado**, pois sua dependência (Job 1) falhou.

Conclusão

Você acaba de proteger seu pipeline. O deploy só acontece se o código, as dependências e a imagem do contêiner passarem nas verificações de segurança.

Exercício

1. Conforme a prática desta aula, replique os passos em seu computador, na sequência:
   - Altere a mensagem “Olá Mundo DevOps” para “Olá Mundo DevSecOps” nos arquivos app.js e app.test.js.
   - Identifique Vulnerabilidades (Localmente): Antes de enviar o código, rode o comando `npm audit` no seu terminal local. Identifique se existem vulnerabilidades.
   - Corrija as Vulnerabilidades: Ainda na sua máquina, execute a correção das vulnerabilidades encontradas:
     - Tente `npm audit fix`.
     - Se necessário, use `npm audit fix --force`.
     - Rode `npm audit` novamente para garantir que a resposta seja "found 0 vulnerabilities".
   - Faça um novo commit na Branch “dev”, o PR e finalmente a merge na main. Verifique se a mensagem foi alterada no servidor (sua maquina).

- Envie o link do repositório criado no github, onde será possível conferir a execução do pipeline CI/CD, juntamente com um pequeno parágrafo explicando em quais situações essa abordagem com DevSecOps pode ser útil dentro de uma empresa.

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional.8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.