15 — Build e deploy

15 — Build e deploy

> Builds por ambiente, o artefato gerado e como ele é servido atrás do ISAPI. A escolha de ambiente acontece **em build** (qual `config.json` é embarcado) e pode ser trocada **em deploy** (substituindo o arquivo) — sem recompilar. **Não há pipeline de CI na referência.** Isso é lacuna, não decisão.

Scripts

Comando	O que faz
`npm run dev`	`env:dev` + Vite dev server (porta 5000)
`npm run start:homolog`	`env:homolog` + dev server apontando para homologação
`npm run build`	`lint` → `env:prod` → `tsc` → `vite build`
`npm run build:homolog`	`env:homolog` → `tsc` → `vite build`
`npm run preview`	Serve o `dist/` localmente

O build de produção inclui o lint

```
"build": "npm run lint && npm run env:prod && tsc && vite build"
```

Quatro etapas, sequenciais, qualquer falha aborta:

1. `lint` — ESLint com zero warnings
2. `env:prod` — gera `public/config.json` a partir de `config/prod.json`
3. `tsc` — type-check completo (`noEmit`)

4. `vite build` — bundle em `dist/`

> `build:homolog` **não roda o lint.** É inconsistente com o `build`. **No projeto novo, inclua o lint nos dois**: `"build:homolog": "npm run lint && npm run env:homolog && tsc && vite build"`.

Por que `tsc` além do `vite-plugin-checker`

O checker do dev só verifica o que foi tocado na sessão. `tsc` verifica o projeto inteiro. Um erro de tipo num arquivo que ninguém abriu passa pelo checker e é pego aqui.

O artefato

```
dist/
├── index.html
├── config.json          ← copiado de public/, define o backend
└── assets/
    ├── index-<hash>.js
    ├── index-<hash>.css
    └── <Pagina>-<hash>.js   ← um chunk por página (lazy)
```

```
build: { outDir: "dist", sourcemap: true }
```

`sourcemap: true` publica os `.map` junto do bundle — qualquer um com acesso à URL lê o código-fonte original. Aceitável em sistema interno; **desligue se o app for exposto na internet**.

Trocar de ambiente sem rebuild

Como o `config.json` é lido em runtime (`fetch("./config.json")` — ver 04), o mesmo `dist/` serve qualquer ambiente:

```
```

```
# aponta um build existente para outro backend
cp config/homolog.json dist/config.json
```

Isso viabiliza promover exatamente o artefato testado em homologação para produção, sem recompilar.

Consequência: o `config.json` do build **não** é a verdade final. Confirme o que está no servidor, não o que foi buildado.

Deploy ISAPI

O backend Curio é servido por `cxIsapiClient.dll` sob IIS. O front é um SPA estático publicado no mesmo IIS, normalmente num diretório virtual ao lado do gateway.

Passos:

1. `npm run build` (ou `build:homolog`)
2. Copiar o conteúdo de `dist/` para o diretório publicado no IIS
3. Ajustar `dist/config.json` se o destino diferir do ambiente do build
4. Validar: abrir a aplicação, fazer login, executar uma tela que chame o backend

Rewrite para SPA

O React Router usa `BrowserRouter` (history API). Uma URL profunda como `/cadastro/fornecedor/incluir` chega ao IIS como um caminho que não existe em disco — e retorna 404.

O IIS precisa de uma regra de rewrite que devolva `index.html` para qualquer caminho que não seja arquivo real:

```
<!-- web.config no diretório publicado -->
<configuration>
  <system.webServer>
    <rewrite>
      <rules>
        <rule name="SPA fallback" stopProcessing="true">
          <match url=".*" />
```

```
          <conditions logicalGrouping="MatchAll">           <add input="{REQUEST_FILENAME}"
matchType="IsFile" negate="true" />           <add input="{REQUEST_FILENAME}"
matchType="IsDirectory" negate="true" />
          </conditions>
          <action type="Rewrite" url="/index.html" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

Sintoma de rewrite ausente: a aplicação funciona navegando pelo menu, mas F5 numa tela interna dá 404.

> É comum uma implementação de referência **não versionar** um `web.config`. Se o projeto novo for publicado em IIS, versione-o junto do código — configuração de servidor não deve viver só na memória de quem publica.

`base` do Vite

Se a aplicação for servida em subcaminho (`https://servidor/sistema/` em vez da raiz), configure:

```
// vite.config.ts
export default defineConfig({ base: "/sistema/" });
```

Sem isso, os `assets/` são requisitados da raiz e não carregam. A referência assume publicação na raiz (`base` não é definido).

CI/CD — o que existe e o que falta

É comum encontrar `.gitlab/` na raiz só com **templates de issue e merge request**:

```
.gitlab/
```

```
├── issue_templates/default.md
└── merge_request_templates/default.md
```

Sem `.gitlab-ci.yml`, não há pipeline. Se o template de MR pede "Pipeline passando" e "Testes escritos ou atualizados" sem que o projeto tenha como cumprir nenhum dos dois, é checklist aspiracional — alinhe o texto ao que existe de verdade.

Recomendação para o projeto novo

Um pipeline mínimo cobre o que hoje depende de disciplina individual:

```
# .gitlab-ci.yml
image: node:22
stages: [verify, build]
cache:
  key: "$CI_COMMIT_REF_SLUG"
  paths: [node_modules/]
verify:
  stage: verify
  script:
    - npm ci
    - npm run lint
    - npx tsc --noEmit
    - npm run format:check
build:
  stage: build
  script:
    - npm ci
    - npm run build
  artifacts:
    paths: [dist/]
    expire_in: 1 week
```

Ajuste os caminhos se o projeto web for submódulo de um monorepo (`cd caminho/do/modulo` antes dos comandos).

Se adotar CI, alinhe o checklist do template de MR ao que o pipeline realmente verifica. Checklist que pede o que não existe treina o time a marcar caixas sem ler.

Antes de publicar

- [ ] `npm run build` passou localmente
- [ ] `config.json` no `dist/` aponta para o ambiente certo
- [ ] Nenhum token de outro ambiente no `config.json` publicado
- [ ] Rewrite de SPA configurado no IIS
- [ ] F5 numa tela interna carrega (valida o rewrite)
- [ ] Login funciona contra o backend do ambiente
- [ ] `sourcemap` condiz com a exposição da aplicação

Revision #2
Created Thu, Aug 20, 2026 5:40 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:54 PM by Geraldo Barbosa