Table of Contents

# 🔄 CÓRTEX BI - Guia de Sincronização com GitHub

**Versão:** 2.0  
**Data:** Outubro 2025  
**Desenvolvido em parceria com:** Manus AI

## 🎯 Objetivo

Configurar seu servidor para rodar o CÓRTEX BI diretamente do GitHub, mantendo tudo sempre atualizado automaticamente.

## 📋 Índice

1. [Método 1: Sincronização Manual Simples](#método-1-sincronização-manual-simples)
2. [Método 2: Script de Auto-Atualização](#método-2-script-de-auto-atualização)
3. [Método 3: Webhook GitHub (Recomendado)](#método-3-webhook-github-recomendado)
4. [Método 4: GitHub Actions CI/CD (Avançado)](#método-4-github-actions-cicd-avançado)
5. [Método 5: Desenvolvimento com Hot Reload](#método-5-desenvolvimento-com-hot-reload)

## 🔧 Método 1: Sincronização Manual Simples

### Configuração Inicial

**1. Clonar o repositório no servidor:**

# No seu servidor (Windows, Linux ou macOS)  
cd /caminho/onde/quer/instalar  
git clone https://github.com/Rimkus85/cortex-bi.git  
cd cortex-bi

**2. Configurar credenciais do Git:**

# Configurar nome e email  
git config --global user.name "Seu Nome"  
git config --global user.email "seu@email.com"  
  
# Configurar autenticação (escolha uma opção)  
  
# Opção A: HTTPS com token  
git config --global credential.helper store  
# Na próxima vez que fizer push, use seu Personal Access Token como senha  
  
# Opção B: SSH (mais seguro)  
ssh-keygen -t ed25519 -C "seu@email.com"  
# Adicione a chave pública (~/.ssh/id\_ed25519.pub) no GitHub  
# Settings > SSH and GPG keys > New SSH key

**3. Atualizar sempre que necessário:**

# Parar o servidor  
./scripts/stop\_ai.sh # ou stop\_ai.bat no Windows  
  
# Puxar últimas alterações  
git pull origin master  
  
# Reinstalar dependências (se houver mudanças)  
pip install -r requirements.txt  
  
# Reiniciar servidor  
./scripts/start\_ai.sh # ou start\_ai.bat no Windows

### Vantagens

✅ Simples e direto  
✅ Controle total sobre quando atualizar  
✅ Não requer configuração adicional

### Desvantagens

❌ Manual - você precisa lembrar de atualizar  
❌ Servidor precisa ser reiniciado manualmente

## 🤖 Método 2: Script de Auto-Atualização

### Criar Script de Atualização Automática

**Linux/macOS** (scripts/auto\_update.sh):

#!/bin/bash  
  
echo "🔄 Iniciando atualização automática do CÓRTEX BI..."  
  
# Diretório do projeto  
PROJECT\_DIR="/caminho/para/cortex-bi"  
cd "$PROJECT\_DIR"  
  
# Verificar se há atualizações  
git fetch origin master  
  
LOCAL=$(git rev-parse HEAD)  
REMOTE=$(git rev-parse origin/master)  
  
if [ "$LOCAL" = "$REMOTE" ]; then  
 echo "✅ Já está na versão mais recente!"  
 exit 0  
fi  
  
echo "📥 Novas atualizações disponíveis. Baixando..."  
  
# Parar servidor  
echo "⏸️ Parando servidor..."  
./scripts/stop\_ai.sh  
  
# Fazer backup da configuração  
echo "💾 Fazendo backup das configurações..."  
cp .env .env.backup  
cp -r config config.backup  
  
# Puxar atualizações  
echo "⬇️ Baixando atualizações..."  
git pull origin master  
  
# Restaurar configurações  
echo "♻️ Restaurando configurações..."  
mv .env.backup .env  
# Mesclar configs se necessário  
  
# Atualizar dependências  
echo "📦 Atualizando dependências..."  
pip install -r requirements.txt --upgrade  
  
# Reiniciar servidor  
echo "🚀 Reiniciando servidor..."  
./scripts/start\_ai.sh  
  
echo "✅ Atualização concluída com sucesso!"

**Windows** (scripts/auto\_update.bat):

@echo off  
echo 🔄 Iniciando atualização automática do CÓRTEX BI...  
  
cd /d C:\caminho\para\cortex-bi  
  
REM Verificar atualizações  
git fetch origin master  
  
for /f %%i in ('git rev-parse HEAD') do set LOCAL=%%i  
for /f %%i in ('git rev-parse origin/master') do set REMOTE=%%i  
  
if "%LOCAL%"=="%REMOTE%" (  
 echo ✅ Já está na versão mais recente!  
 exit /b 0  
)  
  
echo 📥 Novas atualizações disponíveis. Baixando...  
  
REM Parar servidor  
echo ⏸️ Parando servidor...  
call scripts\stop\_ai.bat  
  
REM Backup  
echo 💾 Fazendo backup...  
copy .env .env.backup  
xcopy /E /I config config.backup  
  
REM Atualizar  
echo ⬇️ Baixando atualizações...  
git pull origin master  
  
REM Restaurar config  
echo ♻️ Restaurando configurações...  
copy .env.backup .env  
  
REM Atualizar dependências  
echo 📦 Atualizando dependências...  
pip install -r requirements.txt --upgrade  
  
REM Reiniciar  
echo 🚀 Reiniciando servidor...  
call scripts\start\_ai.bat  
  
echo ✅ Atualização concluída!

### Agendar Execução Automática

**Linux/macOS (cron):**

# Editar crontab  
crontab -e  
  
# Adicionar linha para verificar atualizações a cada hora  
0 \* \* \* \* /caminho/para/cortex-bi/scripts/auto\_update.sh >> /var/log/cortex-bi-update.log 2>&1  
  
# Ou a cada 6 horas  
0 \*/6 \* \* \* /caminho/para/cortex-bi/scripts/auto\_update.sh >> /var/log/cortex-bi-update.log 2>&1  
  
# Ou diariamente às 3h da manhã  
0 3 \* \* \* /caminho/para/cortex-bi/scripts/auto\_update.sh >> /var/log/cortex-bi-update.log 2>&1

**Windows (Task Scheduler):**

# Criar tarefa agendada (executar como Administrador)  
$action = New-ScheduledTaskAction -Execute "C:\caminho\para\cortex-bi\scripts\auto\_update.bat"  
$trigger = New-ScheduledTaskTrigger -Daily -At 3am  
$principal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType ServiceAccount -RunLevel Highest  
Register-ScheduledTask -Action $action -Trigger $trigger -Principal $principal -TaskName "CORTEX\_BI\_Auto\_Update" -Description "Atualização automática do CÓRTEX BI"

### Vantagens

✅ Automático - não precisa lembrar  
✅ Agendável - escolhe quando atualizar  
✅ Faz backup antes de atualizar

### Desvantagens

❌ Atualiza em intervalos fixos (não em tempo real)  
❌ Servidor fica offline durante atualização

## 🎣 Método 3: Webhook GitHub (Recomendado)

Este método atualiza o servidor **imediatamente** quando você faz push no GitHub!

### Passo 1: Criar Endpoint de Webhook

Crie o arquivo src/webhook\_handler.py:

"""  
Webhook Handler para Auto-Deploy do CÓRTEX BI  
Atualiza automaticamente quando há push no GitHub  
"""  
  
from flask import Flask, request, jsonify  
import subprocess  
import hmac  
import hashlib  
import os  
from threading import Thread  
  
app = Flask(\_\_name\_\_)  
  
# Configurações  
WEBHOOK\_SECRET = os.getenv('GITHUB\_WEBHOOK\_SECRET', 'seu-segredo-aqui')  
PROJECT\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
def verify\_signature(payload, signature):  
 """Verifica assinatura do webhook do GitHub"""  
 if not signature:  
 return False  
   
 sha\_name, signature = signature.split('=')  
 if sha\_name != 'sha256':  
 return False  
   
 mac = hmac.new(  
 WEBHOOK\_SECRET.encode(),  
 msg=payload,  
 digestmod=hashlib.sha256  
 )  
   
 return hmac.compare\_digest(mac.hexdigest(), signature)  
  
def update\_and\_restart():  
 """Atualiza código e reinicia servidor"""  
 try:  
 print("🔄 Iniciando atualização...")  
   
 # Ir para diretório do projeto  
 os.chdir(PROJECT\_DIR)  
   
 # Fazer backup das configurações  
 print("💾 Fazendo backup...")  
 subprocess.run(['cp', '.env', '.env.backup'], check=False)  
   
 # Puxar atualizações  
 print("⬇️ Baixando atualizações do GitHub...")  
 result = subprocess.run(['git', 'pull', 'origin', 'master'],   
 capture\_output=True, text=True)  
 print(result.stdout)  
   
 # Restaurar configurações  
 print("♻️ Restaurando configurações...")  
 subprocess.run(['cp', '.env.backup', '.env'], check=False)  
   
 # Atualizar dependências  
 print("📦 Atualizando dependências...")  
 subprocess.run(['pip', 'install', '-r', 'requirements.txt', '--upgrade'],  
 capture\_output=True)  
   
 # Reiniciar servidor  
 print("🚀 Reiniciando servidor...")  
 subprocess.run(['./scripts/stop\_ai.sh'], check=False)  
 subprocess.run(['./scripts/start\_ai.sh'], check=False)  
   
 print("✅ Atualização concluída!")  
   
 except Exception as e:  
 print(f"❌ Erro na atualização: {e}")  
  
@app.route('/webhook/github', methods=['POST'])  
def github\_webhook():  
 """Endpoint para receber webhooks do GitHub"""  
   
 # Verificar assinatura  
 signature = request.headers.get('X-Hub-Signature-256')  
 if not verify\_signature(request.data, signature):  
 return jsonify({'error': 'Invalid signature'}), 401  
   
 # Processar evento  
 event = request.headers.get('X-GitHub-Event')  
 payload = request.json  
   
 if event == 'ping':  
 return jsonify({'message': 'Pong! Webhook configurado com sucesso'}), 200  
   
 if event == 'push':  
 # Verificar se é push na branch master  
 if payload.get('ref') == 'refs/heads/master':  
 print(f"📥 Push recebido: {payload['head\_commit']['message']}")  
   
 # Executar atualização em thread separada  
 thread = Thread(target=update\_and\_restart)  
 thread.start()  
   
 return jsonify({  
 'message': 'Atualização iniciada',  
 'commit': payload['head\_commit']['id']  
 }), 200  
   
 return jsonify({'message': 'Event ignored'}), 200  
  
@app.route('/webhook/status', methods=['GET'])  
def webhook\_status():  
 """Verificar status do webhook"""  
 return jsonify({  
 'status': 'active',  
 'project\_dir': PROJECT\_DIR,  
 'git\_branch': subprocess.run(  
 ['git', 'branch', '--show-current'],  
 capture\_output=True,  
 text=True,  
 cwd=PROJECT\_DIR  
 ).stdout.strip()  
 }), 200  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Rodar em porta separada (não conflitar com CÓRTEX BI)  
 app.run(host='0.0.0.0', port=5001, debug=False)

### Passo 2: Adicionar ao .env

# Webhook GitHub  
GITHUB\_WEBHOOK\_SECRET=seu-segredo-super-secreto-aqui

### Passo 3: Iniciar Servidor de Webhook

**Linux/macOS:**

# Criar script de inicialização  
cat > scripts/start\_webhook.sh << 'EOF'  
#!/bin/bash  
cd /caminho/para/cortex-bi  
nohup python3 src/webhook\_handler.py > logs/webhook.log 2>&1 &  
echo $! > webhook.pid  
echo "✅ Webhook iniciado na porta 5001"  
EOF  
  
chmod +x scripts/start\_webhook.sh  
./scripts/start\_webhook.sh

**Windows:**

REM start\_webhook.bat  
@echo off  
cd C:\caminho\para\cortex-bi  
start /B python src\webhook\_handler.py > logs\webhook.log 2>&1  
echo ✅ Webhook iniciado na porta 5001

### Passo 4: Expor Porta com Ngrok (se servidor local)

Se seu servidor está atrás de firewall/NAT:

# Instalar ngrok  
# https://ngrok.com/download  
  
# Expor porta 5001  
ngrok http 5001  
  
# Copiar URL pública gerada (ex: https://abc123.ngrok.io)

### Passo 5: Configurar Webhook no GitHub

1. Vá para: https://github.com/Rimkus85/cortex-bi/settings/hooks
2. Clique em **“Add webhook”**
3. Preencha:
   * **Payload URL**: http://seu-servidor.com:5001/webhook/github (ou URL do ngrok)
   * **Content type**: application/json
   * **Secret**: O mesmo valor de GITHUB\_WEBHOOK\_SECRET
   * **Which events**: Selecione “Just the push event”
   * **Active**: ✅ Marque
4. Clique em **“Add webhook”**

### Passo 6: Testar

# Fazer um push de teste  
echo "# Teste de webhook" >> README.md  
git add README.md  
git commit -m "Teste de auto-deploy"  
git push origin master  
  
# Verificar logs  
tail -f logs/webhook.log

**Resultado esperado:** - GitHub envia webhook - Servidor recebe notificação - Código é atualizado automaticamente - Servidor reinicia com nova versão

### Vantagens

✅ **Atualização instantânea** - assim que você faz push  
✅ **Automático** - zero intervenção manual  
✅ **Seguro** - assinatura verificada  
✅ **Logs completos** - rastreabilidade

### Desvantagens

❌ Requer porta exposta (5001)  
❌ Servidor fica offline durante atualização (~30s)  
❌ Configuração inicial mais complexa

## 🚀 Método 4: GitHub Actions CI/CD (Avançado)

Deploy automático com zero downtime usando GitHub Actions.

### Passo 1: Criar Workflow

Crie .github/workflows/deploy.yml:

name: Deploy CÓRTEX BI  
  
on:  
 push:  
 branches: [ master ]  
 workflow\_dispatch:  
  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
   
 steps:  
 - name: 📥 Checkout código  
 uses: actions/checkout@v3  
   
 - name: 🐍 Configurar Python  
 uses: actions/setup-python@v4  
 with:  
 python-version: '3.11'  
   
 - name: 📦 Instalar dependências  
 run: |  
 python -m pip install --upgrade pip  
 pip install -r requirements.txt  
   
 - name: 🧪 Executar testes  
 run: |  
 # Adicionar seus testes aqui  
 python -m pytest tests/ || true  
   
 - name: 🚀 Deploy para servidor  
 uses: appleboy/ssh-action@master  
 with:  
 host: ${{ secrets.SERVER\_HOST }}  
 username: ${{ secrets.SERVER\_USER }}  
 key: ${{ secrets.SSH\_PRIVATE\_KEY }}  
 script: |  
 cd /caminho/para/cortex-bi  
 git pull origin master  
 pip install -r requirements.txt --upgrade  
 ./scripts/stop\_ai.sh  
 sleep 2  
 ./scripts/start\_ai.sh  
 echo "✅ Deploy concluído!"

### Passo 2: Configurar Secrets no GitHub

1. Vá para: https://github.com/Rimkus85/cortex-bi/settings/secrets/actions
2. Adicione os secrets:
   * SERVER\_HOST: IP ou domínio do seu servidor
   * SERVER\_USER: Usuário SSH
   * SSH\_PRIVATE\_KEY: Chave privada SSH

### Passo 3: Fazer Push

git add .github/workflows/deploy.yml  
git commit -m "Adicionar CI/CD com GitHub Actions"  
git push origin master

**Resultado:** - Código é testado automaticamente - Deploy é feito no servidor - Servidor reinicia automaticamente

### Vantagens

✅ **CI/CD profissional**  
✅ **Testes automáticos** antes do deploy  
✅ **Histórico de deploys** no GitHub  
✅ **Rollback fácil**

### Desvantagens

❌ Configuração complexa  
❌ Requer acesso SSH ao servidor  
❌ Servidor fica offline durante deploy

## 🔥 Método 5: Desenvolvimento com Hot Reload

Para desenvolvimento ativo, use hot reload (servidor reinicia automaticamente ao detectar mudanças).

### Configuração

**1. Instalar watchdog:**

pip install watchdog

**2. Criar script de desenvolvimento** (scripts/dev.py):

#!/usr/bin/env python3  
"""  
Servidor de desenvolvimento com hot reload  
Reinicia automaticamente quando arquivos são modificados  
"""  
  
import sys  
import time  
import subprocess  
from watchdog.observers import Observer  
from watchdog.events import FileSystemEventHandler  
  
class CodeChangeHandler(FileSystemEventHandler):  
 def \_\_init\_\_(self, restart\_callback):  
 self.restart\_callback = restart\_callback  
 self.last\_restart = 0  
   
 def on\_modified(self, event):  
 if event.src\_path.endswith('.py'):  
 # Evitar múltiplos restarts  
 if time.time() - self.last\_restart > 2:  
 print(f"🔄 Arquivo modificado: {event.src\_path}")  
 self.restart\_callback()  
 self.last\_restart = time.time()  
  
class DevServer:  
 def \_\_init\_\_(self):  
 self.process = None  
   
 def start(self):  
 """Inicia o servidor"""  
 print("🚀 Iniciando CÓRTEX BI...")  
 self.process = subprocess.Popen(  
 [sys.executable, 'src/main\_ai.py'],  
 stdout=subprocess.PIPE,  
 stderr=subprocess.PIPE  
 )  
   
 def stop(self):  
 """Para o servidor"""  
 if self.process:  
 print("⏸️ Parando servidor...")  
 self.process.terminate()  
 self.process.wait()  
   
 def restart(self):  
 """Reinicia o servidor"""  
 self.stop()  
 time.sleep(1)  
 self.start()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 server = DevServer()  
 server.start()  
   
 # Configurar watchdog  
 handler = CodeChangeHandler(server.restart)  
 observer = Observer()  
 observer.schedule(handler, 'src/', recursive=True)  
 observer.start()  
   
 print("👀 Monitorando mudanças em src/...")  
 print("Pressione Ctrl+C para parar")  
   
 try:  
 while True:  
 time.sleep(1)  
 except KeyboardInterrupt:  
 observer.stop()  
 server.stop()  
   
 observer.join()

**3. Executar:**

python scripts/dev.py

**Agora:** - Edite qualquer arquivo .py em src/ - Servidor reinicia automaticamente - Mudanças aplicadas instantaneamente

### Vantagens

✅ **Desenvolvimento rápido**  
✅ **Feedback imediato**  
✅ **Não precisa reiniciar manualmente**

### Desvantagens

❌ Apenas para desenvolvimento  
❌ Não sincroniza com GitHub automaticamente

## 🎯 Comparação dos Métodos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | Automático | Tempo Real | Complexidade | Recomendado Para |
| 1. Manual | ❌ | ❌ | Baixa | Testes iniciais |
| 2. Script Agendado | ✅ | ❌ | Média | Produção simples |
| 3. Webhook | ✅ | ✅ | Média | **Produção (recomendado)** |
| 4. GitHub Actions | ✅ | ✅ | Alta | Empresas/CI/CD |
| 5. Hot Reload | ✅ | ✅ | Baixa | **Desenvolvimento** |

## 💡 Recomendação

### Para Desenvolvimento:

# Use hot reload  
pip install watchdog  
python scripts/dev.py

### Para Produção:

# Use webhook (Método 3)  
python src/webhook\_handler.py &  
# Configure webhook no GitHub

### Para Empresas:

# Use GitHub Actions (Método 4)  
# Configure CI/CD completo com testes

## 🔐 Segurança

### Boas Práticas:

1. **Nunca commite credenciais**

* # Sempre use .env  
  echo ".env" >> .gitignore

1. **Use secrets para webhooks**

* # Gere secret forte  
  openssl rand -hex 32

1. **Restrinja acesso SSH**

* # Use chaves SSH, não senhas  
  ssh-keygen -t ed25519

1. **Firewall**

* # Abra apenas portas necessárias  
  sudo ufw allow 5000/tcp # CÓRTEX BI  
  sudo ufw allow 5001/tcp # Webhook

## 🐛 Troubleshooting

### Webhook não funciona

**Problema:** GitHub não consegue alcançar servidor

**Soluções:** 1. Verificar firewall: bash sudo ufw status sudo ufw allow 5001/tcp

1. Testar localmente:

* curl http://localhost:5001/webhook/status

1. Usar ngrok se atrás de NAT:

* ngrok http 5001

### Git pull falha

**Problema:** Conflitos ou permissões

**Soluções:** 1. Descartar mudanças locais: bash git reset --hard origin/master

1. Verificar permissões:

* sudo chown -R $USER:$USER /caminho/para/cortex-bi

### Servidor não reinicia

**Problema:** Processo travado

**Soluções:** 1. Matar processo: bash pkill -f main\_ai.py

1. Verificar logs:

* tail -f logs/cortexbi.log

## 📞 Resumo

**Melhor opção para você:**

Se quer **atualização automática em tempo real**: → Use **Método 3 (Webhook)** ✅

Se está **desenvolvendo ativamente**: → Use **Método 5 (Hot Reload)** ✅

Se quer **CI/CD profissional**: → Use **Método 4 (GitHub Actions)** ✅

**CÓRTEX BI v2.0** - *Sempre atualizado, sempre disponível* 🔄🚀

Desenvolvido em parceria com **Manus AI** | Outubro 2025