Documentação do BOT System Assist

Visão Geral

O **BOT System Assist** é uma solução em Python desenvolvida para automação de monitoramento e notificação para o time de infraestrutura. Ele utiliza *web scraping* via **Selenium** para acessar uma plataforma web interna, integra-se à **API do Zabbix** para obter métricas e alertas de monitoramento, e utiliza as APIs de **Direct Call** e **Telegram** para remediação de falhas e notificação da equipe. O sistema roda em ambiente Windows, usa ambiente virtual Python (venv) e apresenta rotas HTTP via **Flask** para acionar suas funcionalidades. Em operação, o BOT reduz falhas operacionais ao detectar e notificar problemas automaticamente, garantindo SLA e melhorando o faturamento.

Pré-requisitos

- **Sistema Operacional:** Windows (recomendamos Windows 10 ou superior).
- **Python:** Versão 3.9 ou superior (baixar de python.org e instalar). Flask requer Python ≥3.9[1].
- **Navegador Web e WebDriver:** Chrome ou Firefox + driver compatível (ex: ChromeDriver disponível em sites.google.com/chromium.org/driver[2]). O driver deve estar no PATH do Windows ou no diretório do projeto.
- **Credenciais e Tokens:** Contas/credenciais para:
- Plataforma web interna (usuário e senha).
- **Zabbix:** URL do servidor Zabbix, usuário/API Token (variáveis como ZABBIX_URL, ZABBIX_TOKEN/ZABBIX_USER, ZABBIX_PASSWORD)[3][4].
- **Telegram:** Chave do *Bot Token* fornecida pelo BotFather, e ID do chat/grupo de destino.
- **Direct Call API:** Endpoint e credenciais da API externa (fornecedor ou sistema de telefonia) usada para chamadas diretas de remediação.
- **Ambiente Virtual (venv):** Será criado para instalar dependências sem afetar o sistema global[5][6].

Instalação e Configuração do Ambiente

1. **Instalar Python:** Certifique-se de ter o Python 3.9+ instalado. Durante a instalação, habilite a opção para adicionar Python ao PATH.
2. **Criar Diretório do Projeto:** Em um prompt de comando do Windows (cmd.exe), crie uma pasta para o projeto:
3. `> mkdir C:\Bots\BOT_System_Assist`
4. `> cd C:\Bots\BOT_System_Assist`
5. **Criar ambiente virtual:** No diretório do projeto, execute:
6. `> py -3 -m venv .venv`
   Isso cria um ambiente isolado `.venv`[6].
7. **Ativar ambiente virtual:** Execute:
8. `> .venv\Scripts\activate`
   O prompt mudará para indicar que o venv está ativo[7].
9. **Instalar dependências:** Com o venv ativo, instale as bibliotecas necessárias usando pip. Exemplos de comandos:
10. `pip install selenium` (bindings Selenium para automação de navegador)[8].
11. `pip install Flask` (microframework web)[9].
12. `pip install python-telegram-bot` (cliente da API Telegram)[10].
13. `pip install zabbix_utils` (biblioteca oficial para Zabbix API)[11].
14. Se existir um arquivo *requirements.txt*, também pode usar `pip install -r requirements.txt`.
15. **Baixar/WebDriver:** Baixe o driver do navegador escolhido (ex. ChromeDriver) e coloque-o no PATH ou no diretório do projeto. Isso permite ao Selenium controlar o navegador[12][2].
16. **Configurar variáveis de ambiente:** Em `config.py` ou variáveis de sistema (recomendado no `.env` ou no Windows):
17. `ZABBIX_URL`, `ZABBIX_TOKEN` (ou `ZABBIX_USER`/`ZABBIX_PASSWORD`) para API Zabbix[3][4].
18. `TELEGRAM_BOT_TOKEN` com o token do bot Telegram.
19. `TELEGRAM_CHAT_ID` com o ID do canal ou grupo.
20. `DIRECT_CALL_ENDPOINT` e credenciais da API de chamada direta.
21. Outros: credenciais da plataforma web (por exemplo, `SITE_USER`, `SITE_PASS`, `SITE_URL`).
22. **Clonar/Obter o projeto:** Se disponível no GitHub, clone o repositório ou baixe o ZIP e extraia em `C:\Bots\BOT_System_Assist`. Entre na pasta do projeto:
23. `> cd BOT_System_Assist`
24. **Instalar mais libs se preciso:** Caso o código use outras libs (e.g. `requests`, `schedule`, etc.), instale-as via pip também.

25. **Testar instalação:** Verifique se nenhum erro ocorreu na instalação. Tente executar `python -c "import selenium, flask, telegram, zabbix_utils"` e verifique se não há mensagens de erro.

Estrutura do Projeto

O repositório geralmente contém:
- **`app.py` ou `bot.py`:** Módulo principal que inicia o servidor Flask e/ou orquestra a execução.
- **`routes.py` ou no próprio `app.py`:** Define as rotas HTTP (endpoints) do Flask que acionam funções do BOT.
- **`web_scraping.py`:** Conjunto de funções ou classes que usam Selenium para acessar a plataforma web. Contém métodos como `login_site()`, `coletar_dados()`, etc.
- **`zabbix_api.py`:** Funções de integração com o Zabbix API. Pode usar a classe `ZabbixAPI` de *zabbix_utils*. Métodos típicos: autenticar, consultar problemas (`problem.get`), confirmar ou resolver alertas.
- **`telegram_notify.py`:** Funções para enviar mensagens ao Telegram. Usa `bot.send_message(chat_id, texto)` da biblioteca `python-telegram-bot`. Ex: `notificar_telegram(texto)`.
- **`direct_call.py`:** Funções que acionam a API de chamada direta. Provavelmente usa `requests.post` para enviar comandos ao serviço externo (ex: `realizar_chamada(numero)` ou similar).
- **`config.py`:** Parâmetros de configuração (URLs, tokens, IDs, credenciais).
- **Outros arquivos:** Possíveis scripts auxiliares, arquivos de log, ou pendências.

Execução do Projeto

1. **Iniciar o servidor Flask:** Com o ambiente virtual ativo, rode o módulo principal. Por exemplo:

   ```
   > python app.py
   ```

   Isso iniciará o Flask (por padrão na porta 5000). Confirme no console se o servidor iniciou sem erros.
2. **Permitir tráfego (Firewall):** Caso seja necessário acesso externo às rotas, libere a porta 5000 no firewall do Windows ou utilize um servidor reverso (opcional).
3. **Testar rotas:** Acesse no navegador ou via `curl` as rotas definidas. Por exemplo: `http://localhost:5000/status` deve retornar algo como `{"status": "OK", "last_run": "..."}`.
4. **Agendamento/Monitoramento:** Para automação contínua, pode-se configurar o BOT como um serviço do Windows, ou criar tarefas agendadas que requisitem

as rotas periodicamente. Outra abordagem é utilizar bibliotecas de agendamento em Python (ex: `schedule`, `APScheduler`) dentro do próprio código.

5. **Logs:** O projeto pode gerar logs em arquivo ou console. Verifique as pastas de log ou configurações de `logging` no código para depurar erros.

Configuração das APIs

- **Zabbix API:** Garanta que o usuário e token configurados tenham permissão para ler problemas e executar ações (ex: acknowledge). No código, inicialize a conexão (por exemplo, `api = ZabbixAPI(url=ZABBIX_URL, user=USUARIO, password=SENHA)` ou usando token[4]). Use métodos como `api.problem.get()` para listar problemas críticos. Após operação, pode chamar `api.logout()`.
- **Telegram Bot:** Crie um bot no Telegram via *BotFather*, obtenha o *token*. No código, crie instância do bot (ex: `bot = telegram.Bot(token=TELEGRAM_BOT_TOKEN)`). Para enviar notificação: `bot.send_message(chat_id=CHAT_ID, text="Mensagem de alerta")`. A biblioteca oficial sugere usar `pip install python-telegram-bot`[10].
- **Direct Call API:** Consulte a documentação do provedor dessa API. Normalmente, faz-se uma requisição HTTP (POST) para um endpoint com parâmetros (ex: número de telefone, chave de API). No código, implemente algo como `requests.post(DIRECT_CALL_ENDPOINT, json={...})`. Teste com chamadas de exemplo antes de rodar.
- **Flask Routes:** No arquivo `app.py`, inicializa-se o Flask (`app = Flask(__name__)`). Cada rota é decorada com `@app.route(...)` e implementa uma função. Exemplos de rotas possíveis (ajuste conforme o código real):
- `GET /status` – Retorna um JSON indicando que o BOT está online e dados básicos (e.g. timestamp da última execução).
- `GET /executar_coleta` – Dispara manualmente o processo de scraping e monitoramento. Internamente, chama funções para login e coleta de dados do site, verifica alertas no Zabbix e envia notificações via Telegram.
- `GET /teste_telegram` – Envia uma mensagem de teste via Telegram (útil para verificar configuração).
- `POST /zabbix_callback` – Se o Zabbix estiver configurado para notificar via Webhook, essa rota pode receber a payload (JSON) do alerta e processá-la.
- *Observação:* Ajuste as rotas de acordo com o código real. Cada rota deve documentar: método HTTP, parâmetros esperados, e ação executada.

Lógica Geral de Funcionamento

- **Autenticação no Site (Selenium):** O bot usa o Selenium WebDriver para abrir o navegador e acessar a plataforma web interna. A função `login_site()` preenche o usuário e senha (obtidos de configuração) nos campos de login e faz o login. Pode usar `driver.get(SITE_URL)`, `driver.find_element(By.ID, 'user').send_keys(USER)`, etc. Uma vez logado, o bot navega pelas páginas necessárias.
- **Coleta de Dados:** Após autenticação, o método `coletar_dados()` localiza elementos na página (por exemplo, tabelas de tickets ou status de serviços) e extrai as informações relevantes (e.g. texto, valores). Usa seletores do Selenium (`find_element`, `find_elements`) e armazena os resultados em variáveis ou estruturas (listas/dicionários). Pode fazer filtragens ou cálculos se necessário. Finalmente, fecha o navegador ou página.
- **Integração com Zabbix:** Em paralelo ou após o scraping, o bot acessa a API do Zabbix. Usando a classe `ZabbixAPI` (de `zabbix_utils`), ele executa chamadas como `api.auth()` (ou `api.login(token=...)`) e depois `api.problem.get(output="extend", selectAcknowledges="extend", recent="true", limit=10)` para obter problemas recentes[4]. Se um novo problema crítico é encontrado, o bot pode usar `api.event.acknowledge(eventids, message)` para reconhecer ou sinalizar no Zabbix. Isso reduz o MTTR alertando a equipe imediatamente.
- **Notificação via Telegram:** Cada vez que o bot detecta um evento relevante (pode ser uma nova coleta de dado crítica ou um alerta Zabbix), ele formata uma mensagem informativa. A função `notificar_telegram(mensagem)` usa `bot.send_message(chat_id=CHAT_ID, text=mensagem)` para enviar ao grupo. A mensagem pode incluir detalhes (ex: “Alerta: serviço X fora do ar em [timestamp]”). Isso garante que a equipe seja avisada em tempo real.
- **Chamada de Remediação (Direct Call):** Se um evento requer ação imediata (como reiniciar um serviço ou chamar suporte), o bot executa uma chamada à API de telefonia. A função `realizar_chamada(numero)` faz uma requisição HTTP ao endpoint configurado (ex: `requests.post(DIRECT_CALL_ENDPOINT, json={"number": numero})`). Essa ação automaticamente inicia uma chamada telefônica para o responsável. Garantir tratamento de resposta e erros dessa API (tratando códigos HTTP e retornos).
- **Rotinas e Fluxo de Dados:** Em um ciclo típico, uma rota Flask acionará um fluxo como: **login_site → coletar_dados → verificar_alertas → notificar_telegram → (opcionalmente) realizar_chamada**. Todo o processo deve ter tratamento de

exceções; por exemplo, se o Selenium não encontrar um elemento, deve fazer `driver.quit()` e registrar erro. Sempre há verificações (if/else) para só notificar quando necessário.

Detalhamento das Rotas e Funções

- **Rota `/status` (GET):** Retorna um JSON simples confirmando que o serviço está ativo e a data/hora da última execução bem-sucedida. Não executa ações, apenas para monitoramento do próprio BOT.
- **Rota `/executar` (GET):** Inicia o processo completo de coleta e monitoramento. Internamente, chama funções como `login_site()`, depois `checar_zabbix()`, e finaliza com `notificar_telegram()`. Retorna um relatório básico (ex: “Execução concluída, X novos alertas enviados”).
- **Rota `/teste_telegram` (GET):** Ao ser chamada, dispara o envio de uma mensagem teste ao Telegram (ex: “Teste de conexão OK”). Útil para verificar se as credenciais do bot estão corretas sem rodar todo o fluxo.
- **Rota `/zabbix_webhook` (POST):** Se houver integração via webhook do Zabbix, essa rota recebe dados JSON de um gatilho. O código deve parsear o JSON (e.g. `request.json`) e possivelmente acionar funções de scraping ou notificação baseada nesse alerta específico.
- **Funções principais:**
- `login_site()`: Usa Selenium para abrir o navegador, acessar a URL da plataforma, preencher credenciais e fazer login. Exemplo de passos:

```
driver = webdriver.Chrome()
driver.get(SITE_URL)
driver.find_element(By.ID, "usuario").send_keys(SITE_USER)
driver.find_element(By.ID, "senha").send_keys(SITE_PASS)
driver.find_element(By.ID, "botaoLogin").click()
```

- `coletar_dados()`: Após o login, navega pelas páginas necessárias e coleta dados. Por exemplo, `elementos = driver.find_elements(By.CLASS_NAME, "ticket")` e extrai texto. Fecha o driver no final: `driver.quit()`.
- `conectar_zabbix()`: Inicializa `api = ZabbixAPI(url=ZABBIX_URL, token=ZABBIX_TOKEN)` ou via login[3][4].
- `get_problemas_zabbix()`: Executa `api.problem.get(...)` para obter problemas não reconhecidos. Analisa o retorno JSON para identificar severidades.
- `acknowledge_problema(event_id)`: Chama `api.event.acknowledge(eventids=[event_id], message="Notificado via BOT")` para evitar novas notificações do mesmo evento.

- `notificar_telegram(texto)`: Constrói a mensagem final e chama `bot.send_message(chat_id=CHAT_ID, text=texto, parse_mode="Markdown")`.
- `realizar_chamada(alvo)`: Envia uma requisição à API externa. Por exemplo: `requests.post(DIRECT_CALL_ENDPOINT, headers=auth_header, json={"target": alvo})`. Verifica o status da resposta e registra sucesso ou falha.