Guia Completo de Uso - Codex Cloud Wrapper

📋 Sumário

🔐 Autenticação

Passo 1: Login com conta Nexcode

```
# Login com a conta administrativa
gcloud auth login adm@nexcode.live

# Configurar projeto
gcloud config set project elaihub-prod

# Verificar autenticação
gcloud auth list
```

Passo 2: Obter Token de Identidade

```
# Obter token (válido por ~1 hora)
gcloud auth print-identity-token

# Armazenar em variável para usar em múltiplos comandos
export CLOUD_TOKEN=$(gcloud auth print-identity-token)
echo $CLOUD_TOKEN
```

🧪 Teste com cURL

Exemplo 1: Pergunta Simples

```
# Obter token
export CLOUD_TOKEN=$(gcloud auth print-identity-token)

# Fazer requisição
```

```
curl -X POST https://codex-wrapper-467992722695.us-
central1.run.app/api/v1/exec/stream \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CLOUD_TOKEN" \
  -d '{
    "prompt": "What is 2+2? Answer with just the number.",
    "model": "gpt-4o-mini"
  }' \
  --no-buffer
```

Exemplo 2: Execução de Comando

```
curl -X POST https://codex-wrapper-467992722695.us-
central1.run.app/api/v1/exec/stream \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CLOUD_TOKEN" \
  -d '{
    "prompt": "Create a Python script that prints hello world and execute
it",
    "model": "gpt-4o-mini",
    "timeout_ms": 60000
  }' \
  --no-buffer
```

Exemplo 3: Análise de Arquivo

```
curl -X POST https://codex-wrapper-467992722695.us-
central1.run.app/api/v1/exec/stream \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $CLOUD_TOKEN" \
  -d '{
    "prompt": "List all files in /tmp and show their sizes",
    "model": "gpt-4o-mini"
  }' \
  --no-buffer
```

📤 Upload de Arquivos

Usando gsutil

```
# Upload de arquivo único
gsutil cp meu-arquivo.txt gs://elaistore/uploads/

# Upload de diretório
gsutil -m cp -r meu-diretorio/ gs://elaistore/uploads/
```

```
# Verificar upload
gsutil ls gs://elaistore/uploads/
```

Usando gcloud storage

```
# Upload
gcloud storage cp meu-arquivo.txt gs://elaistore/uploads/

# Upload com metadados
gcloud storage cp meu-arquivo.txt gs://elaistore/uploads/ \
  --content-type=application/json

# Listar arquivos
gcloud storage ls gs://elaistore/uploads/
```

📥 Download do Storage

Download com gsutil

```
# Download arquivo único
gsutil cp gs://elaistore/uploads/meu-arquivo.txt ./

# Download diretório completo
gsutil -m cp -r gs://elaistore/uploads/ ./downloads/

# Download com padrão
gsutil cp gs://elaistore/sessions/*.json ./sessions/
```

Download com gcloud storage

```
# Download arquivo
gcloud storage cp gs://elaistore/uploads/meu-arquivo.txt ./

# Listar e filtrar
gcloud storage ls gs://elaistore/sessions/ --recursive | grep "2025-10-03"
```

🐍 Cliente Python

Instalação de Dependências

```
pip install google-auth requests
```

Script Completo: `codex_cloud_client.py`

```
#!/usr/bin/env python3
"""
Cliente Python para Codex Cloud Wrapper
Autor: Nexcode Team
Data: 2025-10-03
"""

import json
import subprocess
import sys
import time
from typing import Optional, Dict, Any, Generator
import requests

class CodexCloudClient:
    """Cliente para interagir com Codex Cloud Wrapper"""

    def __init__(self, base_url: str = "https://codex-wrapper-
467992722695.us-central1.run.app"):
        self.base_url = base_url
        self.token = self._get_token()

    def _get_token(self) -> str:
        """Obtém token de autenticação do gcloud"""
        try:
            result = subprocess.run(
                ['gcloud', 'auth', 'print-identity-token'],
                capture_output=True,
                text=True,
                check=True
            )
            return result.stdout.strip()
        except subprocess.CalledProcessError as e:
            print(f"Erro ao obter token: {e}", file=sys.stderr)
            print("Execute: gcloud auth login adm@nexcode.live",
file=sys.stderr)
            sys.exit(1)

    def exec_stream(
        self,
        prompt: str,
        model: str = "gpt-4o-mini",
        timeout_ms: int = 60000,
        session_id: Optional[str] = None
    ) -> Generator[Dict[str, Any], None, None]:
        """
        Executa prompt e retorna stream de eventos

        Args:
```

```
            prompt: Instrução para o Codex
            model: Modelo a usar (gpt-4o-mini, gpt-4o, claude-sonnet-4,
etc)
            timeout_ms: Timeout em milissegundos
            session_id: ID da sessão (opcional, será gerado se não
fornecido)

        Yields:
            Dicionários com eventos SSE
        """
        url = f"{self.base_url}/api/v1/exec/stream"
        headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {self.token}"
        }
        payload = {
            "prompt": prompt,
            "model": model,
            "timeout_ms": timeout_ms
        }
        if session_id:
            payload["session_id"] = session_id

        try:
            with requests.post(url, json=payload, headers=headers,
stream=True) as response:
                response.raise_for_status()

                # Parse SSE stream
                for line in response.iter_lines(decode_unicode=True):
                    if not line:
                        continue

                    if line.startswith('event:'):
                        event_type = line[6:].strip()
                    elif line.startswith('data:'):
                        data = line[5:].strip()
                        try:
                            data_json = json.loads(data)
                            yield {
                                'event': event_type if 'event_type' in
locals() else 'unknown',
                                'data': data_json
                            }
                        except json.JSONDecodeError:
                            yield {
                                'event': event_type if 'event_type' in
locals() else 'unknown',
                                'data': data
                            }
        except requests.exceptions.RequestException as e:
            print(f"Erro na requisição: {e}", file=sys.stderr)
            raise
```

```
    def exec_simple(
        self,
        prompt: str,
        model: str = "gpt-4o-mini",
        verbose: bool = False
    ) -> str:
        """
        Executa prompt e retorna apenas a resposta final

        Args:
            prompt: Instrução para o Codex
            model: Modelo a usar
            verbose: Se True, mostra eventos intermediários

        Returns:
            Resposta final do agente
        """
        final_message = ""

        for event in self.exec_stream(prompt, model):
            if verbose:
                print(f"[{event['event']}] {event['data']}")

            # Captura mensagem final
            if event['event'] in ['agent_message', 'agent_output']:
                if isinstance(event['data'], dict) and 'message' in
event['data']:
                    final_message = event['data']['message']
            elif event['event'] == 'task_complete':
                if isinstance(event['data'], dict) and
'last_agent_message' in event['data']:
                    if event['data']['last_agent_message']:
                        final_message = event['data']
['last_agent_message']

        return final_message


def main():
    """Exemplos de uso"""
    client = CodexCloudClient()

    print("=== Exemplo 1: Pergunta Simples ===")
    resposta = client.exec_simple("What is 2+2? Answer with just the
number.")
    print(f"Resposta: {resposta}\n")

    print("=== Exemplo 2: Criar e Executar Script ===")
    resposta = client.exec_simple(
        "Create a Python script that prints 'Hello from Cloud!' and
execute it",
        verbose=True
    )
    print(f"\nResposta final: {resposta}\n")
```

```
    print("=== Exemplo 3: Stream com Eventos ===")
    for event in client.exec_stream("Calculate 5 + 7 and explain the
result"):
        if event['event'] == 'agent_message_delta':
            # Imprime deltas em tempo real
            print(event['data'].get('delta', ''), end='', flush=True)
        elif event['event'] == 'task_complete':
            print("\n[Tarefa concluída]")
            break


if __name__ == "__main__":
    main()
```

Uso do Cliente Python

```
# Dar permissão de execução
chmod +x codex_cloud_client.py

# Executar exemplos
./codex_cloud_client.py

# Ou importar em outro script
python3 -c "
from codex_cloud_client import CodexCloudClient
client = CodexCloudClient()
print(client.exec_simple('List files in /tmp'))
"
```

🟨 Cliente JavaScript/Node.js

Instalação de Dependências

```
npm init -y
npm install node-fetch eventsource
```

Script Completo: `codex-cloud-client.js`

```
#!/usr/bin/env node
/**
 * Cliente JavaScript para Codex Cloud Wrapper
 * Autor: Nexcode Team
 * Data: 2025-10-03
 */
```

```
const { spawn } = require('child_process');
const fetch = require('node-fetch');

class CodexCloudClient {
  constructor(baseUrl = 'https://codex-wrapper-467992722695.us-
central1.run.app') {
    this.baseUrl = baseUrl;
    this.token = null;
  }

  /**
   * Obtém token de autenticação do gcloud
   */
  async getToken() {
    if (this.token) return this.token;

    return new Promise((resolve, reject) => {
      const gcloud = spawn('gcloud', ['auth', 'print-identity-token']);
      let token = '';

      gcloud.stdout.on('data', (data) => {
        token += data.toString();
      });

      gcloud.stderr.on('data', (data) => {
        console.error(`Erro gcloud: ${data}`);
      });

      gcloud.on('close', (code) => {
        if (code !== 0) {
          reject(new Error('Falha ao obter token. Execute: gcloud auth
login adm@nexcode.live'));
        } else {
          this.token = token.trim();
          resolve(this.token);
        }
      });
    });
  }

  /**
   * Executa prompt e retorna stream de eventos
   */
  async execStream(prompt, options = {}) {
    const {
      model = 'gpt-4o-mini',
      timeout_ms = 60000,
      session_id = null,
      onEvent = null
    } = options;

    const token = await this.getToken();
    const url = `${this.baseUrl}/api/v1/exec/stream`;
```

```
    const payload = {
      prompt,
      model,
      timeout_ms
    };
    if (session_id) payload.session_id = session_id;

    const response = await fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`
      },
      body: JSON.stringify(payload)
    });

    if (!response.ok) {
      throw new Error(`HTTP ${response.status}: ${response.statusText}`);
    }

    // Parse SSE stream
    const events = [];
    let currentEvent = null;
    let buffer = '';

    for await (const chunk of response.body) {
      buffer += chunk.toString();
      const lines = buffer.split('\n');
      buffer = lines.pop() || ''; // Mantém última linha incompleta

      for (const line of lines) {
        if (line.startsWith('event:')) {
          currentEvent = line.substring(6).trim();
        } else if (line.startsWith('data:')) {
          const data = line.substring(5).trim();
          let parsedData;

          try {
            parsedData = JSON.parse(data);
          } catch {
            parsedData = data;
          }

          const event = {
            event: currentEvent || 'unknown',
            data: parsedData
          };

          events.push(event);

          if (onEvent) {
            onEvent(event);
          }
```

```
        }
      }
    }

    return events;
  }

  /**
   * Executa prompt e retorna apenas resposta final
   */
  async execSimple(prompt, options = {}) {
    const { verbose = false, model = 'gpt-4o-mini' } = options;
    let finalMessage = '';

    await this.execStream(prompt, {
      model,
      onEvent: (event) => {
        if (verbose) {
          console.log(`[${event.event}]`, event.data);
        }

        // Captura mensagem final
        if (['agent_message', 'agent_output'].includes(event.event)) {
          if (event.data?.message) {
            finalMessage = event.data.message;
          }
        } else if (event.event === 'task_complete') {
          if (event.data?.last_agent_message) {
            finalMessage = event.data.last_agent_message;
          }
        }
      }
    });

    return finalMessage;
  }

  /**
   * Stream em tempo real mostrando deltas
   */
  async execLive(prompt, model = 'gpt-4o-mini') {
    await this.execStream(prompt, {
      model,
      onEvent: (event) => {
        if (event.event === 'agent_message_delta') {
          process.stdout.write(event.data?.delta || '');
        } else if (event.event === 'task_complete') {
          console.log('\n[Tarefa concluída]');
        } else if (event.event === 'error') {
          console.error('\n[Erro]', event.data);
        }
      }
    });
  }
```

```
}

// Exemplos de uso
async function main() {
  const client = new CodexCloudClient();

  console.log('=== Exemplo 1: Pergunta Simples ===');
  const resposta1 = await client.execSimple('What is 2+2? Answer with just
the number.');
  console.log(`Resposta: ${resposta1}\n`);

  console.log('=== Exemplo 2: Criar e Executar Script ===');
  const resposta2 = await client.execSimple(
    "Create a Python script that prints 'Hello from Node.js Client!' and
execute it",
    { verbose: true }
  );
  console.log(`\nResposta final: ${resposta2}\n`);

  console.log('=== Exemplo 3: Stream ao Vivo ===');
  await client.execLive('Calculate 5 + 7 and explain the result');
}

// Executar se chamado diretamente
if (require.main === module) {
  main().catch(console.error);
}

module.exports = CodexCloudClient;
```

Uso do Cliente JavaScript

```
# Dar permissão de execução
chmod +x codex-cloud-client.js

# Executar exemplos
node codex-cloud-client.js

# Ou usar em outro arquivo
# const CodexCloudClient = require('./codex-cloud-client.js');
# const client = new CodexCloudClient();
# client.execSimple('List files in /tmp').then(console.log);
```

🖥️ CLI Dedicado Cloud

Estrutura Proposta

Vou criar um CLI dedicado baseado no `codex-cli` existente, mas configurado exclusivamente para o serviço cloud.

Localização: `/Users/williamduarte/NCMproduto/codex/codex-rs/cloud-cli/`

Funcionalidades:

- ✅ Autenticação automática com gcloud
- ✅ Cache de token
- ✅ Comandos simplificados
- ✅ Upload/download integrado com GCS
- ✅ Histórico de sessões
- ✅ Modo interativo

Comandos Propostos

```
# Executar prompt direto
codex-cloud exec "create a hello world script"

# Modo interativo
codex-cloud interactive

# Upload de arquivo
codex-cloud upload myfile.txt

# Download de sessão
codex-cloud download session-id-123

# Listar sessões
codex-cloud sessions list

# Ver logs de sessão
codex-cloud sessions logs session-id-123

# Configuração
codex-cloud config set model gpt-4o
codex-cloud config set timeout 120000
```

Deseja que eu implemente este CLI dedicado agora?

📚 Referências

URLs do Serviço

- **API Base**: `https://codex-wrapper-467992722695.us-central1.run.app`
- **Endpoint Exec**: `/api/v1/exec/stream`
- **Storage Bucket**: `gs://elaistore/`

Modelos Disponíveis

- `gpt-4o-mini` (padrão, mais rápido)
- `gpt-4o` (mais capaz)

- `claude-sonnet-4` (requer ANTHROPIC_API_KEY)
- `gpt-5` (se disponível)

Limites

- **Timeout padrão**: 30 segundos
- **Timeout máximo**: 300 segundos (5 minutos)
- **Memória**: 2GB
- **CPU**: 1 vCPU

Troubleshooting

Erro 401 Unauthorized:

```
# Token expirado, renovar
unset CLOUD_TOKEN
export CLOUD_TOKEN=$(gcloud auth print-identity-token)
```

Erro 403 Forbidden:

```
# Verificar se está logado com a conta correta
gcloud auth list
# Deve mostrar: adm@nexcode.live
```

Timeout:

```
# Aumentar timeout na requisição
curl ... -d '{"prompt": "...", "timeout_ms": 120000}'
```

🔒 Segurança

⚠ **IMPORTANTE**:

- Nunca commitar tokens no git
- Tokens expiram em ~1 hora
- Use variáveis de ambiente para tokens
- API keys devem estar apenas no Cloud Run, não nos clientes

Boas Práticas

```
# ✅ BOM: Token em variável
export CLOUD_TOKEN=$(gcloud auth print-identity-token)
curl -H "Authorization: Bearer $CLOUD_TOKEN" ...
```

```
# ❌ RUIM: Token hardcoded
curl -H "Authorization: Bearer eyJhbGc..." ...
```

📞 Suporte

- **Documentação Técnica**: `/Users/williamduarte/NCMproduto/codex/docs/wrapper-cloud-run.md`
- **Issues**: Reportar problemas na equipe Nexcode
- **Email**: adm@nexcode.live

Última Atualização: 2025-10-03 **Versão**: 1.0.0 **Status**: ✅ Produção