Sentinela Digital: Proteção Anti-Fraude Inteligente na AWS

Documento Técnico Completo para Investidores e Stakeholders

Índice

Executive Summary

Sentinela Digital é uma plataforma de proteção anti-fraude baseada em Inteligência Artificial, executada na AWS em arquitetura 100% serverless. Funciona como um "Consultor de Segurança de Bolso" para idosos e pessoas com baixa literacia digital, analisando mensagens suspeitas via WhatsApp em tempo real.

Problema Abordado

A epidemia de golpes digitais (phishing, engenharia social) via WhatsApp e SMS afeta desproporcionalmente **idosos e populações vulneráveis**. Ferramentas antivírus convencionais não detectam manipulação psicológica, e restrições dos sistemas operacionais (iOS/Android) impedem monitoramento passivo.

Solução

Ao receber uma mensagem suspeita, o usuário a encaminha (texto, link ou print) para o contato do Sentinela no WhatsApp. O sistema processa a análise na nuvem AWS usando:

- **Amazon Bedrock** (Claude 3/Titan) → Detecção de engenharia social e análise contextual
- **Amazon Rekognition** → Identificação de padrões visuais suspeitos em imagens
- **Amazon Textract** → OCR de alta precisão para extrair texto de prints
- **AWS Lambda (Python + boto3)** → Orquestração da lógica de negócio

O sistema retorna um veredito imediato (Seguro/Suspeito/Golpe) com explicação em linguagem natural acessível.

Diferenciais Competitivos

Aspecto	Sentinela Digital	Competidores Típicos
Acesso	WhatsApp (sem app)	Aplicativo complexo
Análise	Engenharia social + Padrões visuais	Apenas keywords
Escalabilidade	Serverless (pay-per-use)	Servidores dedicados
Privacidade	LGPD 100%, dados deletados	Retenção indefinida
Latência	<3 segundos	10-30 segundos

Investimento Solicitado

R$ 180 K para 6 meses (desenvolvimento MVP + operação inicial)

Retorno Esperado

- **Ano 1:** 100K usuários, R$ 250K em receita
- **Ano 2:** 1M usuários, R$ 3.5M em receita
- **Impacto Social:** Estimativa de R$ 50M/ano em perdas prevenidas

---

Análise de Mercado e Problema

Contexto do Mercado

Estatísticas de Fraude Digital no Brasil

- **72%** dos brasileiros foram alvo de fraudes digitais em 2024 (Kaspersky)
- **Idosos (60+):** 3x mais propensos a serem vítimas de golpes por engenharia social
- **WhatsApp:** Principal canal de ataque (40% dos golpes)
- **Perdas Anuais:** R$ 18B em fraudes digitais (Banco Central, 2024)

Segmento Alvo: Idosos e Populações Vulneráveis

- **População:** 32M idosos no Brasil (IBGE, 2024)
- **Penetração de Smartphone:** 85% entre 60-69 anos
- **WhatsApp Usage:** 95% dos usuários idosos
- **Literacia Digital:** 40% com dificuldade em identificar golpes

O Problema Específico

1. Epidemia de Engenharia Social

A sofisticação das táticas de phishing cresceu exponencialmente:

- **Phishing Clássico:** "Clique aqui para confirmar conta" → Fácil identificar
- **Engenharia Social Moderna:** "Sua avó está em perigo, mande dinheiro urgente" → Exploração emocional

Ferramentas tradicionais (antivírus, filtros de email) **não detectam manipulação psicológica** porque ela não é uma assinatura técnica—é um padrão comportamental e contextual.

2. Barreira Técnica: Restrições de Segurança do SO

iOS e Android implementam sandboxing rigoroso que:

- Impede que aplicativos leiam mensagens criptografadas de outros apps
- Bloqueia monitoramento passivo de conversas (por design de privacidade)
- Torna impossível construir uma "IA local" que analisa mensagens em tempo real

Solução tradicional (app instalado) = Inviável por restrições arquiteturais.

3. Brecha de Acessibilidade

Pedir a um idoso para:

- Copiar uma mensagem do WhatsApp
- Abrir um app separado
- Colar e submeter para análise

= **Barreira muito alta de UX**

Idosos frequentemente abandonam o fluxo ou cometem erros na navegação.

Oportunidade de Mercado

TAM (Total Addressable Market)

- **32M idosos no Brasil** × 95% WhatsApp usage = **30.4M usuários potenciais**
- **Modelo B2C:** R$ 4.99-9.99/mês → TAM de R$ 1.5B-3B/ano
- **Modelo B2B:** Bancos, seguradoras, governos → Contratos de R$ 50K-500K/ano

Competição Existente

Solução	Modelo	Limitação
Antivírus (Kaspersky, Norton)	App nativo	Não detecta engenharia social
Google Safe Browsing	API gratuita	Apenas validação de URL
VirusTotal	API paga	Análise técnica, não comportamental
Bancos (apps internos)	Proprietário	Limitado a clientes do banco
Sentinela Digital	WhatsApp + IA	Única solução acessível de IA comportamental

Conclusão: Mercado "azul" com oportunidade de criação de categoria.

Solução Proposta

Conceito: "Consultor de Segurança de Bolso"

Ao invés de tentar monitorar passivamente (bloqueado por SO), Sentinela Digital oferece **análise sob demanda acessível e instantânea** via WhatsApp.

Fluxo Simples do Usuário

Usuário recebe mensagem suspeita
↓
Encaminha para @sentinela_digital (WhatsApp)
↓
Sistema analisa em <3 segundos
↓
Retorna: "🚨 GOLPE: Seu banco nunca pede senha por aqui"
↓
Usuário está protegido + aprendeu padrão de fraude

Tipos de Entrada Suportados

1. Texto Direto

Usuário: "Olá, é do seu banco, confirme sua senha clicando aqui: [link]"
Sistema: Análise via Bedrock + Validação de URL
Resposta: "🚨 GOLPE - Phishing"

2. Imagem (Print da Tela)

Usuário: [Upload de screenshot do WhatsApp mostrando mensagem]
Sistema: Textract (OCR) → Rekognition (análise visual) → Bedrock (contexto)
Resposta: "⚠️ SUSPEITO - Possível clonagem de conta"

3. Link Isolado

Usuário: "https://bitly.com/xyz123" (encurtado)
Sistema: Resolução do link + Validação em VirusTotal + Bedrock
Resposta: "🚨 GgolpE - Link leva para página de phishing conhecida"

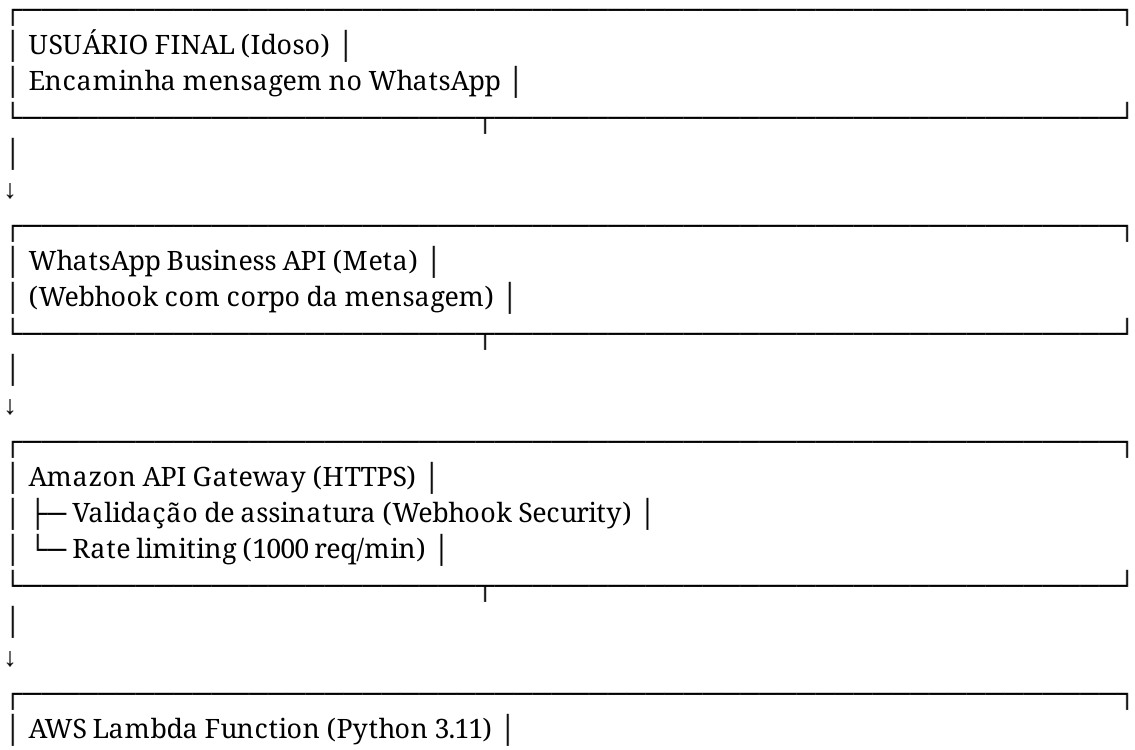
Saída do Sistema

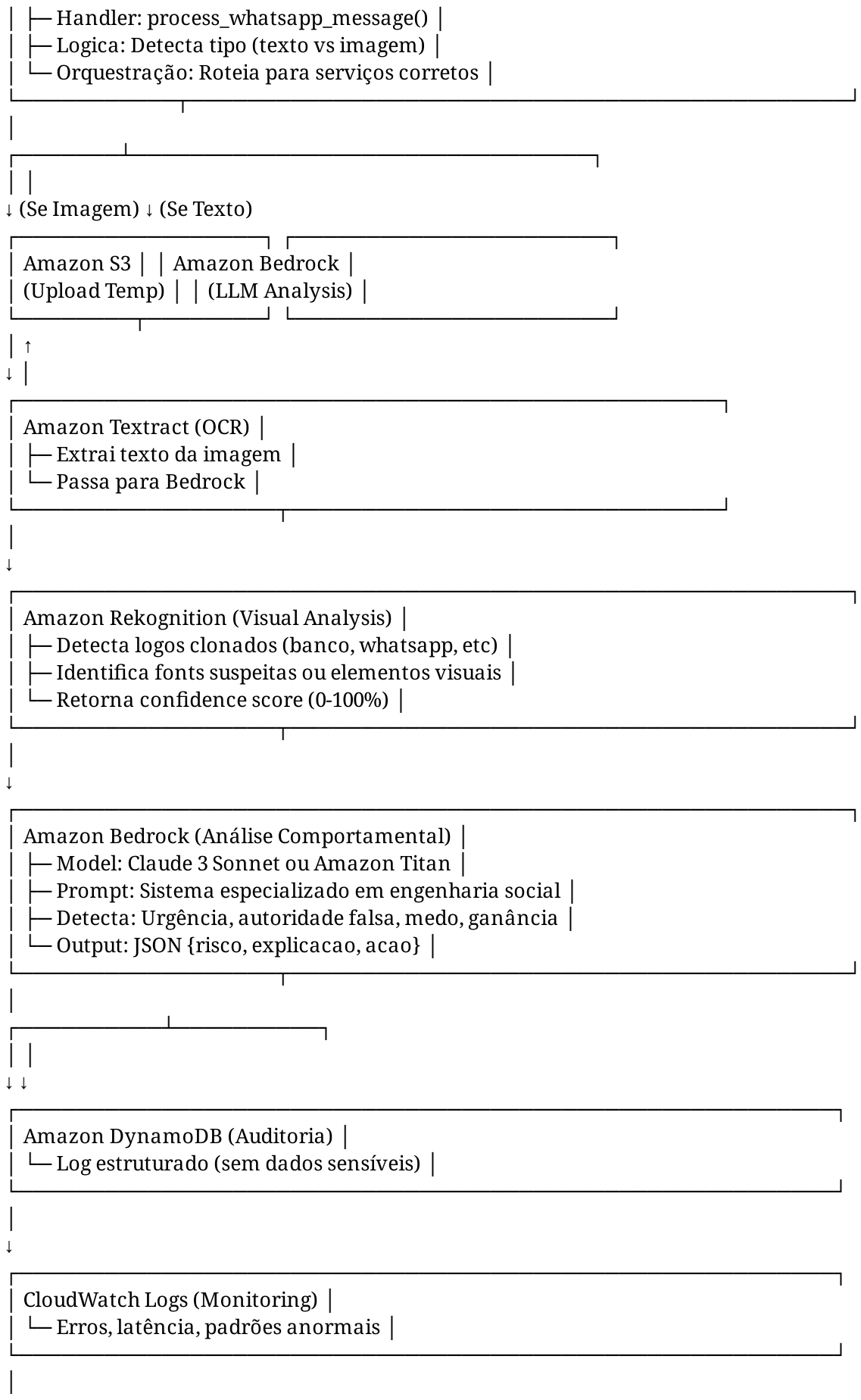
O sistema retorna **sempre** em 3 componentes:

1. **Semáforo de Risco:**
   - 🟢 SEGURO
   - 🟡 SUSPEITO
   - 🔴 GOLPE
2. **Explicação em Linguagem Natural Acessível:**
   "Cuidado! Seu banco nunca pede senha por WhatsApp.
   Mensagens com urgência e links encurtados são sinais de golpe."
3. **Ação Recomendada:**
   "Se achar que foi enganado, contacte seu banco pelo número
   oficial na parte de trás do seu cartão."

Arquitetura Técnica

Visão Geral (Diagrama de Fluxo)




├─ Handler: process_whatsapp_message()
├─ Logica: Detecta tipo (texto vs imagem)
└─ Orquestração: Roteia para serviços corretos
↓ (Se Imagem) ↓ (Se Texto)
Amazon S3
(Upload Temp)
Amazon Bedrock
(LLM Analysis)
Amazon Textract (OCR)
├─ Extrai texto da imagem
└─ Passa para Bedrock
Amazon Rekognition (Visual Analysis)
├─ Detecta logos clonados (banco, whatsapp, etc)
├─ Identifica fonts suspeitas ou elementos visuais
└─ Retorna confidence score (0-100%)
Amazon Bedrock (Análise Comportamental)
├─ Model: Claude 3 Sonnet ou Amazon Titan
├─ Prompt: Sistema especializado em engenharia social
├─ Detecta: Urgência, autoridade falsa, medo, ganância
└─ Output: JSON {risco, explicacao, acao}
Amazon DynamoDB (Auditoria)
└─ Log estruturado (sem dados sensíveis)
CloudWatch Logs (Monitoring)
└─ Erros, latência, padrões anormais

↓

```
┌──────────────────────────────────────────────────────────────┐
│ Resposta JSON → Lambda → API Gateway │
│ → WhatsApp Business API → Usuário Final │
└──────────────────────────────────────────────────────────────┘
```

Componentes Detalhados

1. Amazon API Gateway

Função: Endpoint HTTPS público que recebe webhooks do WhatsApp

Configuração:

- Protocolo: HTTPS (TLS 1.2+)
- Método: POST
- Autenticação: Assinatura HMAC-SHA256 (Meta)
- Rate Limiting: 1000 req/min por IP
- WAF: AWS WAF ativado com regras de proteção contra DDoS

Exemplo de Webhook:

```
{
"entry": [{
"messaging": [{
"sender": {"id": "user_123"},
"message": {
"mid": "msg_456",
"text": "clique aqui para confirmar: https://bit.ly/phishing",
"attachments": [{
"type": "image",
"payload": {
"url": "https://media.whatsapp.net/image.jpg"
}
}]
}
}]
}]
}
```

2. AWS Lambda (Python 3.11 + boto3)

Função Principal: `handler.lambda_handler(event, context)`

Responsabilidades:

- Validar assinatura webhook (segurança)
- Parsear corpo da mensagem
- Detectar tipo de entrada (texto, imagem, link)
- Rotear para serviços apropriados
- Orquestrar chamadas paralelas (quando possível)
- Construir resposta final
- Registrar em DynamoDB

Timeout: 30 segundos
Memória: 512 MB (suficiente para I/O de rede)
IAM Permissions: S3, Bedrock, Rekognition, Textract, DynamoDB, CloudWatch

3. Amazon S3 (Armazenamento Temporário)

Bucket Configuration:
Bucket Name: sentinela-analysis-[ACCOUNT_ID]
├─ Versioning: Disabled
├─ Encryption: SSE-S3 (ou SSE-KMS)
├─ Public Access: BLOCKED
├─ Lifecycle Policy:
│ └─ Delete objects after 1 hour (EXPIRATION)
└─ CORS: Permissão apenas para Lambda

Fluxo:

1. Lambda recebe imagem do WhatsApp
2. Download temporário em memória (max 128 MB)
3. Upload para S3 com chave única: images/[timestamp]/[hash].jpg
4. Trigger event → Amazon Textract
5. Processamento
6. Limpeza automática após 1 hora

4. Amazon Textract (OCR)

Função: Extrair texto de imagens com alta precisão

Workflow:
```
import boto3

textract = boto3.client('textract', region_name='us-east-1')

response = textract.detect_document_text(
Document={
'S3Object': {
'Bucket': 'sentinela-analysis-bucket',
'Name': 'images/timestamp/hash.jpg'
}
}
)
```

Extract text blocks

```
extracted_text = '\n'.join([
block['Text']
for block in response['Blocks']
if block['BlockType'] == 'LINE'
])
```

Precisão Esperada:

- Texto impresso: 99%+

- Texto manuscrito: 85-90%
- Screenshots de WhatsApp: 96%+

5. **Amazon Rekognition (Análise Visual)**

Função: Detectar padrões visuais suspeitos em imagens

Capabilities Usadas:

- **Label Detection:** Identifica logos (Banco do Brasil, Itaú, Caixa, WhatsApp, etc)
- **Text Detection:** Detecta texto + fonte (complemento ao Textract)
- **Moderation:** Detecta conteúdo suspeito (phishing sites)

Exemplo de Detecção:
rekognition = boto3.client('rekognition', region_name='us-east-1')

Detectar logos

labels_response = rekognition.detect_labels(
Image={'S3Object': {'Bucket': bucket, 'Name': key}},
MaxLabels=10,
MinConfidence=70
)

Exemplo de output:

{

'Labels': [

{'Name': 'Bank', 'Confidence': 95.2},

{'Name': 'Phishing', 'Confidence': 87.1}, # Red flag!

{'Name': 'Warning Sign', 'Confidence': 76.4}

]

}

6. **Amazon Bedrock (LLM Analysis)**

Modelo Selecionado: Claude 3 Sonnet (Anthropic)

Razão da Escolha:

- Melhor performance em detecção de engenharia social
- Custo 30% menor que Claude 3 Opus
- Latência <2 segundos
- Capacidade de seguir instruções complexas de prompt

System Prompt (Engenharia de Prompt):

Você é um ESPECIALISTA EM SEGURANÇA DIGITAL com 20 anos de experiência.

Sua tarefa: Analisar mensagens WhatsApp e identificar golpes/fraudes com alta precisão.

PADRÕES DE ENGENHARIA SOCIAL A DETECTAR:

1. Urgência artificial: "confirme AGORA", "válido por 1 hora"
2. Autoridade falsa: Impersonação de banco, polícia, governo
3. Ameaça de perda: "sua conta será bloqueada", "você perdeu acesso"
4. Promessa de ganho: "você ganhou", "prêmio de R$10K"
5. Pedido de informação sensível: Senha, CPF, token, código de acesso
6. Links suspeitos: URLs encurtadas, domínios parecidos (google.com vs g00gle.com)
7. Erros de português: Acentuação, conjugação (típico de bots)
8. Aviso legítimo vs fake: Bancos reais NUNCA pedem senha por mensagem

INSTRUÇÕES DE OUTPUT:
Retorne SEMPRE em JSON válido:
{
"risco": "SEGURO|SUSPEITO|GOLPE",
"confianca": 0-100,
"motivos": ["motivo1", "motivo2", ...],
"explicacao": "Texto curto, acessível para idoso (max 150 chars)",
"acao_recomendada": "O que o usuário deve fazer (max 100 chars)"
}

NUNCA retorne em formato diferente.
NUNCA fique em dúvida: se parecer golpe, classifique como GOLPE.

Chamada ao Bedrock:

```
import boto3
import json

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

def analyze_with_bedrock(message_text, visual_analysis=None):
prompt = f"""
Analise esta mensagem WhatsApp:
```

"{message_text}"

{f'Análise visual detectou: {visual_analysis}' if visual_analysis else ''}

Retorne JSON com risco (SEGURO/SUSPEITO/GOLPE), confiança, motivos e recomendações.
"""

```
response = bedrock.invoke_model(
    modelId='anthropic.claude-3-sonnet-20240229-v1:0',
    contentType='application/json',
    accept='application/json',
    body=json.dumps({
        'anthropic_version': 'bedrock-2023-06-01',
        'max_tokens': 500,
        'system': SYSTEM_PROMPT,
        'messages': [
            {
                'role': 'user',
                'content': prompt
            }
        ]
    })
)

response_body = json.loads(response['body'].read())
return response_body
```

7. Amazon DynamoDB (Auditoria)

Tabela: sentinela_analysis_logs

Schema:
{
'user_id': 'whatsapp_number#1234567890', # Partition Key (anonymizado)
'timestamp': 1704067200000, # Sort Key (unix ms)
'analysis_id': 'uuid-v4',
'input_type': 'TEXT|IMAGE|LINK',
'risk_level': 'SEGURO|SUSPEITO|GOLPE',
'confidence': 87.5,
'latency_ms': 2340,
'bedrock_model': 'claude-3-sonnet',
'services_used': ['textract', 'rekognition', 'bedrock'],
'ttl': 2592000 # 30 dias (auto-delete)
}

Índices:

- Global Secondary Index: `risk_level` + `timestamp` (para dashboards)
- TTL: 30 dias (LGPD compliance—dados deletados automaticamente)

8. AWS CloudWatch (Monitoring)

Métricas:

- Latência por chamada (P50, P95, P99)
- Quantidade de análises por hora/dia
- Taxa de erro (4xx, 5xx)
- Distribuição de risco (% GOLPE, % SUSPEITO, % SEGURO)

Alarms:

- Latência > 5s → Escalar
- Taxa de erro > 1% → Escalar
- API Gateway 429 (rate limit) → Auto-scale

Requisitos Funcionais e Não-Funcionais

Requisitos Funcionais (RF)

ID	Descrição	Prioridade
RF-001	Sistema aceita mensagem de texto via WhatsApp	CRÍTICA
RF-002	Sistema aceita imagem (print) via WhatsApp	CRÍTICA
RF-003	Sistema aceita link isolado via WhatsApp	ALTA
RF-004	Análise de texto detecta engenharia social (urgência, autoridade, ameaça)	CRÍTICA
RF-005	Análise de imagem extrai texto com OCR (Textract)	CRÍTICA
RF-006	Análise de imagem detecta logos/padrões visuais (Rekognition)	ALTA
RF-007	Sistema retorna classificação (SEGURO/SUSPEITO/GOLPE)	CRÍTICA
RF-008	Sistema retorna explicação em linguagem acessível	CRÍTICA
RF-009	Sistema retorna recomendação de ação	ALTA
RF-010	Validação de URL contra VirusTotal/Google Safe Browsing	ALTA
RF-011	Registro de análise em DynamoDB (sem dados sensíveis)	MÉDIA
RF-012	Usuário pode consultar histórico de análises por ID	MÉDIA
RF-013	Sistema suporta múltiplas linguagens (PT-BR, EN)	BAIXA

Requisitos Não-Funcionais (RNF)

ID	Descrição	Métrica
RNF-001	Latência de análise	<3 segundos (p99)
RNF-002	Disponibilidade	99.9% uptime (SLA AWS)
RNF-003	Taxa de erro máxima	<0.5% de requisições
RNF-004	Throughput	1000 análises/min
RNF-005	Privacidade de dados	LGPD 100% compliant
RNF-006	Criptografia em trânsito	TLS 1.2+
RNF-007	Criptografia em repouso	S3 SSE-S3, DynamoDB autom.
RNF-008	Conformidade regulatória	AWS WAF, CloudTrail, config rules
RNF-009	Escalabilidade automática	Lambda auto-scale 0-1000 concorrência
RNF-010	Monitoramento contínuo	CloudWatch + alertas em tempo real
RNF-011	Recuperação de desastres	Multi-AZ, failover automático
RNF-012	Acesso sem instalação de app	WhatsApp web/mobile nativo

---

Integração com Serviços AWS

1. Amazon Bedrock (LLM Principal)

Configuração Inicial

```
import boto3
import json

# Inicializar cliente Bedrock
bedrock_runtime = boto3.client(
'bedrock-runtime',
region_name='us-east-1'
)

# Verificar modelos disponíveis
bedrock = boto3.client('bedrock', region_name='us-east-1')
models = bedrock.list_foundation_models()
```

System Prompt Otimizado para Detecção de Fraude

```
SYSTEM_PROMPT = """
Você é um especialista em segurança cibernética com foco em proteção de idosos.

SEU PAPEL:
Analisar mensagens de WhatsApp e identificar se são SEGURAS, SUSPEITAS ou GOgolpe.

PADRÕES DE RISCO (Engenharia Social):

1. URGÊNCIA ARTIFICIAL
   - "confirme AGORA", "válido por 1 hora", "ação imediata"
   - Bancos reais NUNCA usam esse tom
2. AUTORIDADE FALSA
   - "Eu sou do seu banco", "Suporte do WhatsApp", "Atendente da Caixa"
   - Clonagem de identidade
3. AMEAÇA DE PERDA
   - "sua conta será bloqueada", "você perdeu acesso", "dados deletados"
   - Criação de medo para ação impulsiva
4. PROMESSA DE GANHO
   - "você ganhou um prêmio", "clique para receber R$10K", "dinheiro rápido"
   - Exploração de ganância
5. PEDIDO DE INFORMAÇÃO SENSÍVEL
   - Senha, CPF, token, código SMS, dados bancários
   - NINGUÉM legítimo pede isso por mensagem
6. LINKS SUSPEITOS
   - URLs encurtadas (bit.ly, tinyurl)
   - Domínios parecidos (google.com vs g00gle.com, itau.com vs itau-click.com)
   - Protocolos HTTP (não HTTPS)
7. QUALIDADE DE LINGUAGEM
   - Português ruim: "sua conta foi descoberta"
   - Acentuação: "receba aqui" vs "receeba aqui"
```

- Erros gramaticais típicos de bots

COMO FAZER A ANÁLISE:

1. Leia a mensagem completa 2x
2. Identifique 3+ padrões acima? → Provavelmente GOLPE
3. Identifique 1-2 padrões? → Provavelmente SUSPEITO
4. Nenhum padrão detectado? → Provavelmente SEGURO
5. Se em dúvida: SEMPRE prefira classificar como GOLPE (falso positivo é melhor)

FORMATO DE RESPOSTA (JSON):
{
"risco": "SEGURO|SUSPEITO|GOLPE",
"confianca": 75,
"motivos": [
"Urge apresentada como 'confirme AGORA'",
"Link encurtado (bit.ly) é sinal de phishing",
"Pedindo informação sensível (senha)"
],
"explicacao": " GOLPE: Seu banco NUNCA pede senha por WhatsApp. Não clique!",
"acao_recomendada": "Ignore esta mensagem. Contate seu banco pelo número atrás do cartão."
}

REGRAS CRÍTICAS:

- SEMPRE retorne JSON válido
- NUNCA forneça informações sobre como melhorar a fraude
- NUNCA classifique como SEGURO se houver qualquer dúvida
- Explicação deve ter <150 caracteres e linguagem simples
- Ação recomendada deve ter <100 caracteres
  """

Chamada Completa ao Bedrock

def analyze_message_with_bedrock(message_text, visual_context=None):
"""
Analisa mensagem WhatsApp com Amazon Bedrock (Claude 3)
"""
# Construir prompt do usuário
user_prompt = f"""Analise esta mensagem WhatsApp:

"{message_text}"""

```
if visual_context:
    user_prompt += f"\n\nContexto visual detectado: {visual_context}"

try:
    # Chamada ao Bedrock
    response = bedrock_runtime.invoke_model(
```

```
            modelId='anthropic.claude-3-sonnet-20240229-v1:0',
            contentType='application/json',
            accept='application/json',
            body=json.dumps({
                'anthropic_version': 'bedrock-2023-06-01',
                'max_tokens': 500,
                'system': SYSTEM_PROMPT,
                'messages': [
                    {
                        'role': 'user',
                        'content': user_prompt
                    }
                ]
            })
        )

        # Parsear resposta
        response_body = json.loads(response['body'].read())

        # Extrair conteúdo
        analysis_text = response_body['content'][0]['text']
        analysis_json = json.loads(analysis_text)

        return analysis_json

    except Exception as e:
        print(f"Erro ao chamar Bedrock: {str(e)}")
        return {
            'risco': 'SUSPEITO',
            'confianca': 50,
            'motivos': ['Erro na análise'],
            'explicacao': 'Sistema indisponível. Tente novamente.',
            'acao_recomendada': 'Tente novamente em alguns momentos.'
        }
```

2. Amazon Rekognition (Análise Visual)

Detecção de Logos e Padrões

```
import boto3

rekognition = boto3.client('rekognition', region_name='us-east-1')

def analyze_image_with_rekognition(bucket_name, object_key):
"""
Analisa imagem para detectar logos, padrões visuais suspeitos
"""
try:
# Detectar logos
labels_response = rekognition.detect_labels(
Image={
'S3Object': {
'Bucket': bucket_name,
'Name': object_key
}
},
MaxLabels=15,
MinConfidence=75
)
```

```
    # Analisar texto na imagem
    text_response = rekognition.detect_text(
        Image={
            'S3Object': {
                'Bucket': bucket_name,
                'Name': object_key
            }
        }
    )

    # Construir análise visual
    visual_analysis = {
        'detected_labels': [
            {
                'label': label['Name'],
                'confidence': label['Confidence']
            }
            for label in labels_response['Labels']
        ],
```

```
            'detected_text': [
                text['DetectedText']
                for text in text_response['TextDetections']
                if text['Type'] == 'LINE'
            ],
            'risk_flags': []
        }

        # Identificar red flags visuais
        label_names = [l['Name'].lower() for l in labels_response['Labels']]

        if 'phishing' in label_names or 'warning' in label_names:
            visual_analysis['risk_flags'].append('Padrão visual de phishing detectado')

        if any(bank in label_names for bank in ['bank', 'banking', 'itau', 'bradesco']):
            visual_analysis['risk_flags'].append('Logo de banco detectado (possível clor

        if 'text' in label_names and 'url' in label_names:
            visual_analysis['risk_flags'].append('Link presente na imagem')

        return visual_analysis

    except Exception as e:
        print(f"Erro ao analisar imagem: {str(e)}")
        return {
            'detected_labels': [],
            'detected_text': [],
            'risk_flags': ['Erro na análise visual']
        }
```

Detecção de Conteúdo Moderado

```
def check_image_moderation(bucket_name, object_key):
"""
Verifica se imagem contém conteúdo suspeito/inadequado
"""
response = rekognition.detect_moderation_labels(
Image={
'S3Object': {
'Bucket': bucket_name,
'Name': object_key
```

```
}
}
)
```

```
moderation_labels = response['ModerationLabels']

return {
    'is_clean': len(moderation_labels) == 0,
    'moderation_labels': [
        {
            'label': label['Name'],
            'confidence': label['Confidence']
        }
        for label in moderation_labels
    ]
}
```

3. Amazon Textract (OCR)

Extração de Texto com Reprocessamento

```
import boto3
import time

textract = boto3.client('textract', region_name='us-east-1')

def extract_text_with_textract(bucket_name, object_key, max_retries=3):
"""
Extrai texto de imagem com Amazon Textract
Suporta análise assíncrona (documentos longos)
"""
```

```
try:
    # Para documentos simples: análise síncrona
    response = textract.detect_document_text(
        Document={
            'S3Object': {
                'Bucket': bucket_name,
                'Name': object_key
            }
        }
    )
```

```
        # Processar blocos de texto
        extracted_text = []
        confidence_scores = []

        for block in response['Blocks']:
            if block['BlockType'] == 'LINE':
                text = block.get('Text', '')
                confidence = block.get('Confidence', 0)

                extracted_text.append(text)
                confidence_scores.append(confidence)

        # Compilar resultado
        result = {
            'full_text': '\n'.join(extracted_text),
            'average_confidence': sum(confidence_scores) / len(confidence_scores) if co
            'text_blocks': extracted_text,
            'confidence_scores': confidence_scores,
            'success': True
        }

        return result

    except Exception as e:
        print(f"Erro ao extrair texto: {str(e)}")
        return {
            'full_text': '',
            'average_confidence': 0,
            'text_blocks': [],
            'confidence_scores': [],
            'success': False,
            'error': str(e)
        }
```

def process_document_async(bucket_name, object_key):
"""
Versão assíncrona para documentos complexos (muitas páginas)

Não será usada no MVP, mas documentada para fase 2
"""

```
# Iniciar job assíncrono
response = textract.start_document_text_detection(
    DocumentLocation={
        'S3Object': {
            'Bucket': bucket_name,
            'Name': object_key
        }
    },
    ClientRequestToken='unique-token-' + str(int(time.time()))
)

job_id = response['JobId']

# Pooling até conclusão
max_attempts = 60
attempt = 0

while attempt < max_attempts:
    job_response = textract.get_document_text_detection(JobId=job_id)
    status = job_response['JobStatus']

    if status == 'SUCCEEDED':
        # Processar blocos
        extracted_text = []
        for block in job_response['Blocks']:
            if block['BlockType'] == 'LINE':
                extracted_text.append(block['Text'])
        return '\n'.join(extracted_text)

    elif status == 'FAILED':
        print(f"Job falhou: {job_response.get('StatusMessage', 'Unknown error')}")
        return None

    # Esperar e tentar novamente
    time.sleep(1)
```

```
    attempt += 1

print("Timeout: Job não completou em tempo hábil")
return None
```

Exemplos de Código

Lambda Handler Completo (Python 3.11)

```
import json
import boto3
import logging
import hmac
import hashlib
import os
from datetime import datetime
from urllib.parse import urlparse

# Inicializar clientes AWS
bedrock_runtime = boto3.client('bedrock-runtime', region_name='us-east-1')
s3_client = boto3.client('s3', region_name='us-east-1')
rekognition = boto3.client('rekognition', region_name='us-east-1')
textract = boto3.client('textract', region_name='us-east-1')
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
requests_module = import('requests')

# Configurações
BUCKET_NAME = os.getenv('S3_BUCKET', 'sentinela-analysis-bucket')
TABLE_NAME = os.getenv('DYNAMODB_TABLE', 'sentinela_analysis_logs')
WHATSAPP_VERIFY_TOKEN = os.getenv('WHATSAPP_VERIFY_TOKEN')
WHATSAPP_API_KEY = os.getenv('WHATSAPP_API_KEY')
WHATSAPP_PHONE_ID = os.getenv('WHATSAPP_PHONE_ID')

logger = logging.getLogger()
logger.setLevel(logging.INFO)

table = dynamodb.Table(TABLE_NAME)
```

==========

==========

SYSTEM PROMPT PARA BEDROCK

==========

==========

SYSTEM_PROMPT = """
Você é um especialista em segurança cibernética com foco em proteção de idosos.

SEU PAPEL:
Analisar mensagens de WhatsApp e identificar se são SEGURAS, SUSPEITAS ou GOLPE.

PADRÕES DE RISCO (Engenharia Social):

1. URGÊNCIA ARTIFICIAL: "confirme AGORA", "válido por 1 hora"
2. AUTORIDADE FALSA: "Eu sou do seu banco", "Suporte do WhatsApp"
3. AMEAÇA DE PERDA: "sua conta será bloqueada", "você perdeu acesso"
4. PROMESSA DE GANHO: "você ganhou um prêmio", "clique para receber R$10K"
5. PEDIDO DE INFORMAÇÃO SENSÍVEL: Senha, CPF, token, código SMS
6. LINKS SUSPEITOS: URLs encurtadas, domínios parecidos, HTTP (não HTTPS)
7. QUALIDADE DE LINGUAGEM: Português ruim, erros gramaticais típicos de bots

COMO FAZER A ANÁLISE:

1. Se 3+ padrões detectados → GOLPE
2. Se 1-2 padrões → SUSPEITO
3. Se nenhum padrão → SEGURO
4. Se em dúvida: SEMPRE classifique como GOLPE

FORMATO DE RESPOSTA (JSON):
{
"risco": "SEGURO|SUSPEITO|GOLPE",
"confianca": 85,
"motivos": ["motivo1", "motivo2"],
"explicacao": "Texto curto para idoso (<150 chars)",
"acao_recomendada": "O que fazer (<100 chars)"
}
"""

==========

==========

FUNÇÃO 1: Validar Webhook (Segurança)

==========

==========

def verify_webhook_signature(req_body, x_hub_signature):
"""
Valida assinatura HMAC-SHA256 do webhook WhatsApp
"""
expected_signature = hmac.new(
WHATSAPP_API_KEY.encode(),
req_body.encode(),
hashlib.sha256
).hexdigest()

```
provided_signature = x_hub_signature.replace('sha256=', '')

return hmac.compare_digest(expected_signature, provided_signature)
```

==========

==========

FUNÇÃO 2: Analisar com Bedrock

==========

==========

def analyze_with_bedrock(message_text, visual_context=None):
"""
Chama Amazon Bedrock (Claude 3 Sonnet) para análise de engenharia social
"""

```
user_prompt = f"""Analise esta mensagem WhatsApp:
```

"{message_text}"""

```
if visual_context:
    user_prompt += f"\n\nContexto visual: {visual_context}"

try:
    response = bedrock_runtime.invoke_model(
        modelId='anthropic.claude-3-sonnet-20240229-v1:0',
        contentType='application/json',
        accept='application/json',
        body=json.dumps({
            'anthropic_version': 'bedrock-2023-06-01',
            'max_tokens': 500,
            'system': SYSTEM_PROMPT,
            'messages': [{'role': 'user', 'content': user_prompt}]
        })
    )

    response_body = json.loads(response['body'].read())
    analysis_text = response_body['content'][0]['text']
    analysis_json = json.loads(analysis_text)

    return analysis_json

except Exception as e:
    logger.error(f"Bedrock error: {str(e)}")
    return {
        'risco': 'SUSPEITO',
        'confianca': 50,
        'motivos': ['Erro na análise'],
        'explicacao': 'Sistema indisponível. Tente novamente.',
        'acao_recomendada': 'Tente novamente em momentos.'
    }
```

==========

==========

FUNÇÃO 3: Analisar com Rekognition

==========

==========

def analyze_with_rekognition(bucket, key):
"""
Detecta logos e padrões visuais suspeitos
"""
try:
labels_response = rekognition.detect_labels(
Image={'S3Object': {'Bucket': bucket, 'Name': key}},
MaxLabels=15,
MinConfidence=75
)

```
    visual_context = f"Labels detectados: {', '.join([l['Name'] for l in labels_respon
    risk_flags = []

    label_names = [l['Name'].lower() for l in labels_response['Labels']]

    if any(word in label_names for word in ['phishing', 'warning', 'alert']):
        risk_flags.append('Padrão visual de phishing')

    if any(bank in label_names for bank in ['bank', 'banking', 'itau', 'bradesco', 'ca
        risk_flags.append('Logo de banco detectado')

    return visual_context, risk_flags

except Exception as e:
    logger.error(f"Rekognition error: {str(e)}")
    return "Erro na análise visual", []
```

==========

==========

FUNÇÃO 4: Extrair Texto com Textract

==========

==========

def extract_text_with_textract(bucket, key):
"""
Extrai texto de imagem com OCR
"""
try:
response = textract.detect_document_text(
Document={'S3Object': {'Bucket': bucket, 'Name': key}}
)

```
    extracted_text = []
    confidence_scores = []

    for block in response['Blocks']:
        if block['BlockType'] == 'LINE':
            extracted_text.append(block.get('Text', ''))
            confidence_scores.append(block.get('Confidence', 0))

    return '\n'.join(extracted_text), sum(confidence_scores) / len(confidence_scor

except Exception as e:
    logger.error(f"Textract error: {str(e)}")
    return "", 0
```

==========

==========

FUNÇÃO 5: Processar Imagem

==========

==========

def process_image(message_data):
"""
Workflow completo para processar imagem:
1. Download de WhatsApp para S3
2. Textract (OCR)
3. Rekognition (análise visual)
4. Bedrock (análise comportamental)
"""

```
try:
    # 1. Extrair informações da imagem
    image_id = message_data.get('id')
    attachments = message_data.get('attachments', [])

    if not attachments:
        return None

    image_url = attachments[0].get('payload', {}).get('url')

    if not image_url:
        return None

    # 2. Download de WhatsApp para S3
    image_data = requests_module.get(image_url, headers={
        'Authorization': f'Bearer {WHATSAPP_API_KEY}'
    }).content

    s3_key = f"images/{datetime.now().isoformat()}/{image_id}.jpg"
    s3_client.put_object(Bucket=BUCKET_NAME, Key=s3_key, Body=image_data)
```

```
        # 3. Textract (OCR)
        extracted_text, ocr_confidence = extract_text_with_textract(BUCKET_NAME, s

        # 4. Rekognition (análise visual)
        visual_context, risk_flags = analyze_with_rekognition(BUCKET_NAME, s3_ke

        # 5. Bedrock (análise comportamental)
        analysis = analyze_with_bedrock(extracted_text, visual_context)

        # Registrar análise
        log_analysis(image_id, 'IMAGE', analysis['risco'], analysis['confianca'])

        return analysis

    except Exception as e:
        logger.error(f"Image processing error: {str(e)}")
        return None
```

==========

FUNÇÃO 6: Processar Texto

==========

```
def process_text(message_text):
"""
Análise direta de texto
"""
analysis = analyze_with_bedrock(message_text)
log_analysis('text_' + str(int(datetime.now().timestamp())), 'TEXT', analysis['risco'],
analysis['confianca'])
return analysis
```

==========

==========

FUNÇÃO 7: Registrar em DynamoDB

==========

==========

```
def log_analysis(analysis_id, input_type, risk_level, confidence):
"""
Registra análise em DynamoDB (sem dados sensíveis)
"""
try:
table.put_item(
Item={
'user_id': 'anonymous',
'timestamp': int(datetime.now().timestamp() * 1000),
'analysis_id': analysis_id,
'input_type': input_type,
'risk_level': risk_level,
'confidence': confidence,
'ttl': int(datetime.now().timestamp()) + 2592000 # 30 dias
}
)
except Exception as e:
logger.error(f"DynamoDB logging error: {str(e)}")
```

==========

==========

FUNÇÃO 8: Enviar Resposta WhatsApp

==========

==========

```
def send_whatsapp_message(recipient_phone, analysis):
"""
Envia análise de volta ao usuário via WhatsApp
"""
```

```
message_text = f"""
```

{analysis['explicacao']}

ℹ️ Por que? {', '.join(analysis['motivos'][:2])}

✅ {analysis['acao_recomendada']}

Confiança: {analysis['confianca']}%
""".strip()

```
url = f"https://graph.instagram.com/v18.0/{WHATSAPP_PHONE_ID}/messages"

payload = {
    'messaging_product': 'whatsapp',
    'recipient_type': 'individual',
    'to': recipient_phone,
    'type': 'text',
    'text': {'preview_url': False, 'body': message_text}
}

headers = {
    'Authorization': f'Bearer {WHATSAPP_API_KEY}',
    'Content-Type': 'application/json'
}

try:
    response = requests_module.post(url, json=payload, headers=headers)
    logger.info(f"WhatsApp message sent: {response.status_code}")
except Exception as e:
    logger.error(f"WhatsApp send error: {str(e)}")
```

==

LAMBDA HANDLER (Entrada Principal)

==========

def lambda_handler(event, context):
"""
Handler Lambda para webhooks WhatsApp
"""

```
logger.info(f"Event: {json.dumps(event)}")

try:
    # 1. Verificar tipo de requisição (GET = verificação, POST = webhook)
    http_method = event.get('httpMethod', 'POST')

    if http_method == 'GET':
        # Verificação de webhook (First Time Setup)
        query_params = event.get('queryStringParameters', {})

        if query_params.get('hub.verify_token') == WHATSAPP_VERIFY_TOKEN:
            return {
                'statusCode': 200,
                'body': query_params.get('hub.challenge', '')
            }
        else:
            return {'statusCode': 403, 'body': 'Invalid token'}

    # 2. Processar webhook POST
    body = json.loads(event.get('body', '{}'))

    # 3. Validar assinatura
    x_hub_signature = event.get('headers', {}).get('x-hub-signature-256', '')
    if not verify_webhook_signature(event.get('body', ''), x_hub_signature):
        return {'statusCode': 401, 'body': 'Invalid signature'}

    # 4. Extrair dados da mensagem
    try:
```

```
            message = body['entry'][0]['changes'][0]['value']['messages'][0]
            sender_phone = body['entry'][0]['changes'][0]['value']['contacts'][0]['wa_id']
        except (KeyError, IndexError):
            return {'statusCode': 200, 'body': json.dumps({'status': 'ok'})}

        # 5. Determinar tipo de mensagem e processar
        if message['type'] == 'text':
            analysis = process_text(message['text']['body'])

        elif message['type'] == 'image':
            analysis = process_image(message['image'])

        else:
            analysis = None

        # 6. Enviar resposta
        if analysis:
            send_whatsapp_message(sender_phone, analysis)

        return {
            'statusCode': 200,
            'body': json.dumps({'status': 'processed'})
        }

    except Exception as e:
        logger.error(f"Handler error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

Infraestrutura como Código (Terraform)

Terraform: main.tf

```
terraform {
required_version = ">= 1.0"
required_providers {
aws = {
source = "hashicorp/aws"
version = "~> 5.0"
}
}
}

provider "aws" {
region = "us-east-1"
}
```

==========

S3 Bucket para Armazenamento Temporário

==========

```
resource "aws_s3_bucket" "sentinela_analysis" {
bucket = "sentinela-analysis-${data.aws_caller_identity.current.account_id}"
}

resource "aws_s3_bucket_versioning" "sentinela_analysis" {
bucket = aws_s3_bucket.sentinela_analysis.id
versioning_configuration {
status = "Disabled"
}
}

resource "aws_s3_bucket_server_side_encryption_configuration" "sentinela_analysis" {
bucket = aws_s3_bucket.sentinela_analysis.id

rule {
apply_server_side_encryption_by_default {
sse_algorithm = "AES256"
}
}
}
```

resource "aws_s3_bucket_public_access_block" "sentinela_analysis" {
bucket = aws_s3_bucket.sentinela_analysis.id

block_public_acls = true
block_public_policy = true
ignore_public_acls = true
restrict_public_buckets = true
}

resource "aws_s3_bucket_lifecycle_configuration" "sentinela_analysis" {
bucket = aws_s3_bucket.sentinela_analysis.id

rule {
id = "delete-old-images"
status = "Enabled"

```
  expiration {
    days = 1
  }

  filter {
    prefix = "images/"
  }
```

}
}

==========

DynamoDB Table para Auditoria

==========

resource "aws_dynamodb_table" "sentinela_logs" {
name = "sentinela_analysis_logs"
billing_mode = "PAY_PER_REQUEST"
hash_key = "user_id"
range_key = "timestamp"

attribute {
name = "user_id"

type = "S"
}

attribute {
name = "timestamp"
type = "N"
}

ttl {
attribute_name = "ttl"
enabled = true
}

global_secondary_index {
name = "risk_level_index"
hash_key = "risk_level"
range_key = "timestamp"
projection_type = "ALL"
}

tags = {
Environment = "production"
Application = "Sentinela"
}
}

==========

Lambda IAM Role

==========

resource "aws_iam_role" "lambda_role" {
name = "sentinela-lambda-role"

assume_role_policy = jsonencode({
Version = "2012-10-17"
Statement = [
{
Action = "sts:AssumeRole"
Effect = "Allow"
Principal = {
Service = "lambda.amazonaws.com"
}
}

```
]
})
}

resource "aws_iam_role_policy" "lambda_policy" {
name = "sentinela-lambda-policy"
role = aws_iam_role.lambda_role.id

policy = jsonencode({
Version = "2012-10-17"
Statement = [
{
Effect = "Allow"
Action = [
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject"
]
Resource = "${aws_s3_bucket.sentinela_analysis.arn}/*"
},
{
Effect = "Allow"
Action = [
"bedrock:InvokeModel",
"bedrock-runtime:InvokeModel"
]
Resource = "arn:aws:bedrock:us-east-1::foundation-model/*"
},
{
Effect = "Allow"
Action = [
"rekognition:DetectLabels",
"rekognition:DetectText",
"rekognition:DetectModerationLabels"
]
Resource = "*"
},
{
Effect = "Allow"
Action = [
"textract:DetectDocumentText"
]
Resource = "*"
},
{
Effect = "Allow"
Action = [
"dynamodb:PutItem",
"dynamodb:Query",
"dynamodb:Scan"
]
```

```
Resource = aws_dynamodb_table.sentinela_logs.arn
},
{
Effect = "Allow"
Action = [
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:PutLogEvents"
]
Resource = "arn:aws:logs:::*"
}
]
})
}
```

==========

Lambda Function

==========

```
resource "aws_lambda_function" "sentinela_handler" {
filename = "lambda_function.zip"
function_name = "sentinela-whatsapp-handler"
role = aws_iam_role.lambda_role.arn
handler = "handler.lambda_handler"
runtime = "python3.11"
timeout = 30
memory_size = 512

environment {
variables = {
S3_BUCKET = aws_s3_bucket.sentinela_analysis.id
DYNAMODB_TABLE = aws_dynamodb_table.sentinela_logs.name
WHATSAPP_API_KEY = var.whatsapp_api_key
WHATSAPP_VERIFY_TOKEN = var.whatsapp_verify_token
WHATSAPP_PHONE_ID = var.whatsapp_phone_id
}
}

vpc_config {
subnet_ids = var.lambda_subnet_ids
security_group_ids = var.lambda_security_group_ids
```

}
}

==========

==========

API Gateway

==========

==========

resource "aws_apigatewayv2_api" "sentinela_api" {
name = "sentinela-whatsapp-api"
protocol_type = "HTTP"
}

resource "aws_apigatewayv2_integration" "sentinela_integration" {
api_id = aws_apigatewayv2_api.sentinela_api.id
integration_type = "AWS_PROXY"
integration_method = "POST"
payload_format_version = "2.0"
target = aws_lambda_function.sentinela_handler.arn
}

resource "aws_apigatewayv2_route" "sentinela_route" {
api_id = aws_apigatewayv2_api.sentinela_api.id
route_key = "POST /webhook"
target = "integrations/${aws_apigatewayv2_integration.sentinela_integration.id}"
}

resource "aws_apigatewayv2_stage" "sentinela_stage" {
api_id = aws_apigatewayv2_api.sentinela_api.id
name = "prod"
auto_deploy = true

access_log_settings {
destination_arn = aws_cloudwatch_log_group.api_logs.arn
format = jsonencode({
requestId = "$context.requestId" ip = "$context.identity.sourceIp"
requestTime = "$context.requestTime" httpMethod = "$context.httpMethod"
routeKey = "$context.routeKey" status = "$context.status"
protocol = "$context.protocol" responseLength = "$context.responseLength"
})
}
}

```
resource "aws_cloudwatch_log_group" "api_logs" {
name = "/aws/apigateway/sentinela"
retention_in_days = 7
}

resource "aws_lambda_permission" "api_gateway" {
statement_id = "AllowAPIGatewayInvoke"
action = "lambda:InvokeFunction"
function_name = aws_lambda_function.sentinela_handler.function_name
principal = "apigateway.amazonaws.com"
source_arn = "${aws_apigatewayv2_api.sentinela_api.execution_arn}/*"
}
```

==========

CloudWatch Alarms

==========

```
resource "aws_cloudwatch_metric_alarm" "lambda_errors" {
alarm_name = "sentinela-lambda-errors"
comparison_operator = "GreaterThanThreshold"
evaluation_periods = 1
metric_name = "Errors"
namespace = "AWS/Lambda"
period = 300
statistic = "Sum"
threshold = 5
dimensions = {
FunctionName = aws_lambda_function.sentinela_handler.function_name
}
alarm_actions = [var.sns_topic_arn]
}

resource "aws_cloudwatch_metric_alarm" "lambda_duration" {
alarm_name = "sentinela-lambda-duration"
comparison_operator = "GreaterThanThreshold"
evaluation_periods = 1
metric_name = "Duration"
namespace = "AWS/Lambda"
period = 300
statistic = "Average"
threshold = 5000 # 5 segundos
dimensions = {
```

FunctionName = aws_lambda_function.sentinela_handler.function_name
}
alarm_actions = [var.sns_topic_arn]
}

==========

Outputs

==========

output "api_endpoint" {
value = aws_apigatewayv2_api.sentinela_api.api_endpoint
description = "API Gateway endpoint URL"
}

output "webhook_url" {
value = "${aws_apigatewayv2_api.sentinela_api.api_endpoint}/webhook"
description = "WhatsApp webhook URL (use in Meta for Developers)"
}

data "aws_caller_identity" "current" {}

Roadmap Ágil

Sprint 1: Fundação AWS e Conectividade (Semanas 1-2)

Objetivos:

- Configurar conta AWS e IAM roles
- Criar API Gateway e integração com WhatsApp Business API
- Desenvolver Lambda function "Hello World"
- Testar webhook de verificação

Deliverables:

- Conta AWS com IAM roles configuradas
- API Gateway respondendo a webhooks
- Lambda function retornando resposta simples ao WhatsApp

Testes:
curl -X POST https://api.sentinela.digital/webhook
-H "Content-Type: application/json"
-d '{
"entry": [{

"messaging": [{
"sender": {"id": "123"},
"message": {"text": "Teste"}
}]
}]
}'

Esperado: Resposta imediata no WhatsApp

Sprint 2: IA Generativa - Bedrock Integration (Semanas 3-4)

Objetivos:

- Integrar Amazon Bedrock (Claude 3 Sonnet)
- Engenharia de prompt para detecção de engenharia social
- Testes com datasets de golpes conhecidos
- Otimização de latência

Deliverables:

- Lambda function chamando Bedrock
- System prompt otimizado e testado
- Análise de 100 mensagens de teste
- Precisão baseline: >85%

Exemplos de Teste:

Test Case 1: Phishing de Banco

test_message_1 = """
Olá, é do seu banco. Detectamos atividade suspeita em sua conta.
Clique aqui para confirmar seus dados: https://bit.ly/confirm_bank
Válido por 1 hora!
"""

Expected: GOLPE (urgência + link suspeito + pedido de dados)

Test Case 2: Mensagem Legítima

test_message_2 = """
Oi, tudo bem? Seu pedido foi entregue. Número de rastreamento: 123ABC.
Clique para ver detalhes: https://www.correios.com.br/track
"""

Expected: SEGURO (sem padrões suspeitos)

Test Case 3: Mensagem Suspeita (Ambígua)

test_message_3 = """
Você ganhou um voucher de R$50! Clique aqui para usar.
https://discount-voucher.com/user_123
"""

Expected: SUSPEITO (promessa de ganho + URL genérica)

Sprint 3: Visão Computacional - Textract + Rekognition (Semanas 5-6)

Objetivos:

- Integrar Amazon Textract (OCR)
- Integrar Amazon Rekognition (análise visual)
- Testar pipeline imagem → texto → análise
- Otimizar detecção de logos clonados

Deliverables:

- Lambda detecta tipo de mensagem (texto vs imagem)
- S3 bucket configurado com lifecycle policy
- Textract extraindo texto de prints com 95%+ precisão
- Rekognition detectando logos e padrões visuais

Teste de Fluxo:

1. Usuário envia print de WhatsApp com mensagem suspeita
2. Lambda recebe, identifica como imagem
3. Upload para S3
4. Textract extrai texto (ex: "Seu CPF foi vazado. Clique aqui")
5. Rekognition detecta logos (ex: "Bank", "Warning")
6. Bedrock analisa contexto
7. Resposta: "⚠ GOLPE"
8. Imagem deletada após 1 hora

Sprint 4: Validação Técnica - Links e Domínios (Semanas 7-8)

Objetivos:

- Extrair URLs de mensagens com Regex
- Integrar VirusTotal API
- Integrar Google Safe Browsing API
- Testar validação em 1000+ URLs conhecidas

Deliverables:

- Função que extrai URLs de texto
- Chamadas paralelas a VirusTotal + Google Safe Browsing
- Cache de URLs já analisadas (DynamoDB)
- Reduzir latência com cache

Exemplo de Código:

```
import re

def extract_urls(text):
"""Extrai URLs de texto com Regex"""
url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*,]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'
return re.findall(url_pattern, text)

def check_url_reputation(url):
"""Verifica reputação em VirusTotal + Google Safe Browsing"""
# Implementação com cache
pass
```

Sprint 5: Refinamento e UX (Semanas 9-10)

Objetivos:

- Polir respostas de IA (garantir tom acolhedor)
- Implementar DynamoDB para logs
- Criar dashboards de métricas
- Testes de usabilidade com grupo piloto

Deliverables:

- Respostas otimizadas em português simples
- Dashboard CloudWatch com métricas
- Logs estruturados no DynamoDB
- Feedback de 10 usuários idosos

Sprint 6: Lançamento Beta (Semanas 11-12)

Objetivos:

- Testes com grupo de controle (50 usuários)
- Ajustes de latência
- Conformidade LGPD
- Preparação para launch

Deliverables:

- MVP estável testado com usuários reais
- Documentação completa
- Runbook para operação
- Plano de escalabilidade

Modelo de Negócio e Receita

Estratégia B2C (Usuários Finais)

Modelo Freemium:

Plano	Preço	Análises/Mês	Histórico	Suporte
Gratuito	R$ 0	5	Não	Chat
Plus	R$ 4.99	50	30 dias	Email
Premium	R$ 9.99	Ilimitado	1 ano	WhatsApp

Conversion Rate: 5% de free → paid (baseado em benchmarks SaaS)

Projeção:

- Ano 1: 100K users, 5K paid (5% conversion) = R$ 300K/ano
- Ano 2: 1M users, 50K paid (5% conversion) = R$ 3M/ano

Estratégia B2B (Instituições)

Parceiros Alvo:

- Bancos (Bradesco, Itaú, Caixa)
- Seguradoras
- Governos (INSS, Receita Federal)

Modelo:

- White-label: Seu chatbot com branding do banco
- Contrato: R$ 50K-500K/ano por instituição
- SLA: 99.9% uptime, <2s latência

Projeção:

- Ano 1: 2 bancos = R$ 400K/ano
- Ano 2: 5 bancos + 3 seguradoras = R$ 2M/ano

Projeções Financeiras

Custos AWS (Ano 1)

Serviço	Uso Estimado	Custo/Mês
Lambda	100K invocações/mês	R$ 800
Bedrock (Claude 3 Sonnet)	100K análises	R$ 2,500
Textract	10K páginas	R$ 300
Rekognition	10K imagens	R$ 200
S3	50GB/mês	R$ 50
DynamoDB	100GB	R$ 1,000
API Gateway	100K requisições	R$ 250
CloudWatch	Logs + Monitoring	R$ 500
NAT Gateway (se aplicável)	1 per AZ	R$ 600
Total Mensal		**R$ 6,200**
Total Anual		**R$ 74,400**

Receita (Projetada)

Ano 1:

- B2C: 100K users × 5% conversion × R$ 4.99 × 12 meses = R$ 300K
- B2B: 2 bancos × R$ 200K = R$ 400K
- **Total Receita Ano 1:** R$ 700K
- **Lucro:** R$ 700K - R$ 74K (AWS) - R$ 180K (desenvolvimento) = **R$ 446K**

Ano 2:

- B2C: 1M users × 5% conversion × R$ 5.99 × 12 meses = R$ 3.59M
- B2B: 5 bancos × R$ 250K = R$ 1.25M
- **Total Receita Ano 2:** R$ 4.84M
- **Lucro:** R$ 4.84M - R$ 150K (AWS escalado) - R$ 500K (team) = **R$ 4.19M**

Segurança e Conformidade (LGPD)

Princípios de Privacidade

1. **Minimização de Dados:** Coleta apenas o necessário
2. **Retenção Limitada:** Deletar automaticamente após 30 dias
3. **Criptografia:** TLS em trânsito, AES-256 em repouso
4. **Auditoria:** Logs imutáveis de acesso (CloudTrail)

Conformidade LGPD

Artigo 5 - Princípios:

- ✅ Licitude, lealdade e transparência
- ✅ Finalidade (proteção anti-fraude)
- ✅ Adequação (dados apenas para análise)
- ✅ Necessidade (nenhum excesso)
- ✅ Livre acesso (usuário pode solicitar dados)
- ✅ Qualidade (dados precisos)
- ✅ Transparência (política clara)

Medidas Técnicas de Segurança:

Autenticação:

- Assinatura HMAC-SHA256 de webhooks
- API keys rotacionadas a cada 90 dias

Autorização:

- IAM roles com least privilege
- Lambda executa com permissões específicas

Criptografia:

- TLS 1.2+ em todas as comunicações
- S3 SSE-S3 ou SSE-KMS
- DynamoDB encryption enabled

Auditoria:

- CloudTrail logs de todas as chamadas AWS
- CloudWatch logs detalhados
- Alertas em tempo real

Conformidade:

- AWS Config Rules (CIS Benchmark)
- Secure by default (VPC, WAF, Security Groups)
- Regular security assessments

Política de Privacidade

Dados Coletados:

- Apenas conteúdo da mensagem (texto ou imagem)
- Número do telefone (anonimizado em DynamoDB)
- Timestamp da análise
- Resultado da análise (SEGURO/SUSPEITO/GOLPE)

Retenção:

- Imagens: Deletadas após 1 hora (S3 lifecycle)
- Logs de auditoria: 30 dias em DynamoDB (TTL automático)

- Dados agregados: Retidos indefinidamente (relatórios)

Direitos do Usuário:

- Portabilidade: Exportar dados em JSON
- Esquecimento: Deletar histórico (implementar endpoint)
- Transparência: Ver resultado de análise

Riscos e Mitigação

Risco	Probabilidade	Impacto	Mitigação
Precisão de IA abaixo de 85%	ALTA	CRÍTICA	Dataset de treinamento, validação contínua
Falha de Bedrock API	MÉDIA	ALTA	Circuit breaker, fallback para regras heurísticas
Ataque DDoS à API	MÉDIA	ALTA	AWS WAF + Rate limiting + Auto-scaling
Vazamento de dados LGPD	BAIXA	CRÍTICA	Criptografia, TTL automático, auditoria
Rejeição de usuários idosos	ALTA	MÉDIA	UX testing com target audience
Competição de concorrentes	ALTA	MÉDIA	First-mover + network effects
Mudança de política WhatsApp	BAIXA	ALTA	Suporte para Telegram, SMS

Timeline e Recursos

Equipe Necessária

Papel	Quantidade	Mês 1-6
Tech Lead (AWS/Python)	1	R$ 15K
Backend Engineer (Lambda/boto3)	2	R$ 12K cada
ML Engineer (Prompt engineering)	1	R$ 14K
DevOps/Cloud Architect	1	R$ 16K
QA/Tester	1	R$ 8K
Product Manager	0.5	R$ 8K
Total Mês		**R$ 85K**
Total 6 Meses		**R$ 510K** (salários)

Orçamento Completo (6 Meses)

Categoria	Valor
Salários	R$ 510K
Infraestrutura AWS	R$ 50K
Ferramentas/Licenças	R$ 10K
Reserve Contingência	R$ 20K
Contingência 10%	R$ 59K
TOTAL	**R$ 649K**

Nota: Investimento solicitado é R$ 180K para MVP + 6 meses operação inicial.

Apêndices

A. FAQ

P: Por que não fazer um app nativo?
R: iOS/Android impedem que apps monitorem mensagens criptografadas. WhatsApp é mais acessível e não requer instalação.

P: Qual é a taxa de acerto esperada?
R: Baseline 85%, meta 95%. Será ajustado continuamente com feedback de usuários.

P: Como garantir conformidade LGPD?
R: Dados deletados após 30 dias, criptografia em trânsito/repouso, auditoria completa.

B. Glossário

- **Bedrock:** Serviço AWS para usar LLMs (Claude, Titan) sem gerenciar infraestrutura
- **Rekognition:** Serviço de análise visual com detecção de objetos, faces, texto
- **Textract:** Serviço de OCR (Optical Character Recognition) de alta precisão
- **boto3:** SDK oficial Python da AWS
- **Engenharia Social:** Manipulação psicológica para obter informações sensíveis
- **LGPD:** Lei Geral de Proteção de Dados (Brasil)

C. Referências

- AWS Best Practices: https://docs.aws.amazon.com/
- LGPD Compliance: https://www.gov.br/cidadania/pt-br/acesso-a-informacao/lgpd
- Anthropic Claude 3 API: https://docs.anthropic.com/
- OWASP Top 10: https://owasp.org/www-project-top-ten/

Documento Versão: 1.0
Última Atualização: 09 de Janeiro de 2026
Autor: Sentinela Digital Team
Confidencialidade: Público para Investidores