AI SDR Agent Skill — Full Pipeline

Built by Amir Baldiga using Manus AI

An autonomous AI SDR (Sales Development Representative) pipeline that finds B2B leads, researches them, verifies contact emails, writes personalized outreach, sends via Gmail, and manages follow-ups — all without human intervention.

How It Works

```
Find Lead (Apollo) → Research Company → Find CEO → Verify Email → Write Email → Send → Monitor Replies → Follow-up
```

SKILL.md — The Agent's Playbook

```
---
name: sdr-agent
description: >
  AI SDR agent pipeline for finding, researching, and contacting B2B leads.
  Handles the full flow from Apollo search to Gmail send, plus inbox monitoring
  and follow-up management.
---
```

Phase 1: Find Lead (Apollo API)

Search Apollo for companies matching ICP (Ideal Customer Profile) criteria.

```
POST https://api.apollo.io/v1/mixed_companies/search
Header: X-Api-Key: $APOLLO_API_KEY
```

Default ICP filters (customizable per request):

- **Location**: Israel
- **Company size**: 1-20 employees
- **Tags**: SaaS
- **Funding**: Any amount (min: $1)
- **Exclude**: AI/ML companies, defense, law, government

From results, pick ONE company that fits. Extract: name, website, LinkedIn, industry, employee count, location, description.

Phase 2: Research Company

Run in parallel where possible:

1. **Crunchbase API** — funding rounds, industry, about
2. **Web search** — recent news, product updates, funding, milestones
3. **LinkedIn company page** — team size, recent posts, hiring signals

Identify a **specific opportunity angle** for the pitch. If no good angle exists, go back to Phase 1 and pick another company.

Phase 3: Find Decision Maker

Find CEO, Founder, or Co-Founder. Get:

- Full name
- Title
- LinkedIn URL

Search methods: Apollo people data, web search, LinkedIn.

Phase 4: Find & Verify Email

The email finder pipeline (open source tools, zero cost):

```
python find_email.py --first FIRST --last LAST --domain DOMAIN
```

Steps:

1. **MX record lookup** — identify the company's mail servers
2. **Generate email candidates** — first@, first.last@, f.last@, flast@, etc.
3. **SMTP verification** — check if the mailbox exists (250 OK = confirmed)
4. **Holehe check** — verify if the email is registered on 120+ platforms
5. **Web search fallback** — ContactOut, RocketReach, etc.

Confidence scoring:

Score	Evidence
10	SMTP 250 OK + Holehe hits on multiple platforms
8-9	Multiple web sources agree + Holehe hits
6-7	Single web source or pattern match only
Below 6	Do not proceed — find alternative contact

Phase 5: Write Outreach Email

Style rules (Milo voice — the AI SDR agent):

- **Sender**: Milo, AI SDR Agent
- **Language**: Hebrew with natural English tech terms
- **Tone**: Casual, direct, slightly cynical/funny. Smart friend, not salesperson.
- **Subject line**: Tricky, slang-y, self-aware. NOT corporate or clickbait.
- **Length**: 4-5 short paragraphs max
- **Spacing**: Double line breaks between paragraphs

- **NO pricing/money talk** in the email
- **NO signature block** (appended automatically by the send script)
- **Self-referential humor**: Milo acknowledges being AI, uses it as the pitch
- **Reference SPECIFIC findings** from Phase 2 (product updates, milestones, news)
- **End with low-pressure CTA**
- **Anti-AI writing**: no em dashes (—), no rule of three, no promotional language

Format: HTML with RTL wrapper for Hebrew.

```
<div dir="rtl" style="text-align: right; font-family: Arial, sans-serif;">
  <!-- email content -->
</div>
```

Phase 6: Send

Autonomous mode: Send directly without user approval. Self-verify before sending:

- [ ] Email confidence >= 8
- [ ] Company fits ICP (not AI, not defense, not law, not gov)
- [ ] Email body references specific company findings
- [ ] No pricing/money talk, no AI writing patterns
- [ ] Hebrew RTL format correct

Send via Gmail API with auto-appended signature and token refresh.

After sending, update CRM tracker with the new lead entry.

Monitoring & Follow-ups

Phase 7: Check Replies

The reply checker script:

1. Checks inbox for emails FROM each tracked contact

2. If reply found → updates CRM status to “replied”, saves snippet
3. If no reply and follow-up is due (7 days since last contact) → flags for follow-up
4. Sends daily summary email to the user with all replies

Phase 8: Follow-up Emails

Daily limit: max 5 follow-ups per day.

Follow-up rules:

- Max **3 follow-ups** per lead (after initial email)
- **7 days** between each follow-up
- Same voice but shorter
- Reference the previous email
- Add a **new angle, insight, or question** each time
- After 3 unanswered follow-ups → mark lead as “closed”, move on

Follow-up subject: Reply to original thread (Re: original subject)

CRM Tracker

Simple JSON-based CRM. Each lead record:

```
{
  "company": "Example Corp",
  "domain": "example.com",
  "contact_name": "John Doe",
  "contact_email": "john@example.com",
  "linkedin": "https://linkedin.com/in/johndoe",
  "thread_id": "gmail_thread_id",
  "first_sent": "2026-03-08",
  "status": "sent",
  "follow_ups_sent": 0,
  "max_follow_ups": 3,
  "next_follow_up": "2026-03-15",
  "replied": false,
  "reply_date": null,
  "reply_summary": null
}
```

Scheduled Operations

Schedule	Time (Israel)	Action
Outreach #1	09:00	Find lead → Research → Email → Send
Reply Check	10:00	Check replies + send follow-ups
Outreach #2	11:00	Find lead → Research → Email → Send
Outreach #3	13:00	Find lead → Research → Email → Send
Outreach #4	15:00	Find lead → Research → Email → Send
Outreach #5	17:00	Find lead → Research → Email → Send

Scripts

find_email.py — Email Finder & Verifier

```
#!/usr/bin/env python3
"""Find and verify email for a person at a company domain.
Usage: python find_email.py --first NAME --last NAME --domain DOMAIN
"""
import smtplib, dns.resolver, subprocess, json, argparse, sys

def get_mx_records(domain):
    try:
        answers = dns.resolver.resolve(domain, "MX")
        return [str(r.exchange) for r in answers]
    except Exception:
        return []

def generate_candidates(first, last, domain):
    f, l = first.lower(), last.lower()
    candidates = [
        f"{f}@{domain}", f"{f}.{l}@{domain}", f"{f[0]}.{l}@{domain}",
        f"{f[0]}{l}@{domain}", f"{l}@{domain}", f"{f[0]}{l[0]}@{domain}",
        f"ceo@{domain}", f"info@{domain}", f"contact@{domain}"
    ]
    return list(set(candidates))

def verify_smtp(email, mx_servers):
    for mx in mx_servers:
        try:
            with smtplib.SMTP(mx, 25, timeout=10) as s:
                s.ehlo()
                s.mail("test@example.com")
                code, msg = s.rcpt(email)
                if code == 250:
                    return True, f"250 OK on {mx}"
                elif code == 550:
                    return False, f"550 Not Found on {mx}"
                return None, f"Code {code} on {mx}"
        except Exception as e:
            continue
    return None, "No MX responded"

def run_holehe(email):
    try:
```

```
        r = subprocess.run(["holehe", email], capture_output=True,
text=True, timeout=60)
        lines = [l for l in r.stdout.split("\n") if "[+]" in l]
        return lines
    except Exception:
        return []

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--first", required=True)
    parser.add_argument("--last", required=True)
    parser.add_argument("--domain", required=True)
    args = parser.parse_args()

    print(f"Finding email for {args.first} {args.last} @ {args.domain}")
    mx = get_mx_records(args.domain)
    print(f"MX records: {mx}")

    candidates = generate_candidates(args.first, args.last, args.domain)
    print(f"Generated {len(candidates)} candidates")

    results = {}
    for c in candidates:
        if c.endswith(args.domain):
            ok, msg = verify_smtp(c, mx)
            results[c] = {"smtp": ok, "detail": msg}
            if ok:
                print(f"  VERIFIED: {c} - {msg}")
            else:
                print(f"  {c} - {msg}")

    # Holehe on top candidates
    top = [c for c, r in results.items() if r["smtp"]]
    if not top:
        top = [f"{args.first.lower()}@{args.domain}",
               f"{args.first.lower()}.{args.last.lower()}@{args.domain}",
               f"{args.first.lower()[0]}{args.last.lower()}@{args.domain}"]
    for c in top[:3]:
        hits = run_holehe(c)
        if hits:
            print(f"  Holehe [{c}]: {len(hits)} platforms found")
            results.setdefault(c, {})["holehe"] = hits

    output = {"domain": args.domain, "mx": mx, "candidates": results}
    print(json.dumps(output, indent=2, ensure_ascii=False))
```

send_email.py — Gmail API Sender (HTML + RTL + Auto-Signature)

```
#!/usr/bin/env python3
"""Send HTML email via Gmail API with OAuth credentials.
Usage: python send_email.py --to EMAIL --subject SUBJECT --body-file
HTML_FILE [--dry-run]
"""
import argparse, base64, json, os, requests
from email.mime.text import MIMEText

CREDS_PATH = "/path/to/your/gmail_credentials.json"
CLIENT_SECRET_PATH = "/path/to/your/client_secret.json"
SIGNATURE_PATH = "/path/to/your/signature.html"

def load_tokens():
    with open(CREDS_PATH) as f:
        return json.load(f)

def save_tokens(tokens):
    with open(CREDS_PATH, "w") as f:
        json.dump(tokens, f, indent=2)

def refresh_access_token(tokens):
    with open(CLIENT_SECRET_PATH) as f:
        creds = json.load(f)
    key = "installed" if "installed" in creds else "web"
    r = requests.post("https://oauth2.googleapis.com/token", data={
        "refresh_token": tokens["refresh_token"],
        "client_id": creds[key]["client_id"],
        "client_secret": creds[key]["client_secret"],
        "grant_type": "refresh_token"
    })
    if r.status_code == 200:
        tokens["access_token"] = r.json()["access_token"]
        save_tokens(tokens)
        return tokens
    raise RuntimeError(f"Token refresh failed: {r.text}")

def get_valid_token():
    tokens = load_tokens()
    r = requests.get("https://www.googleapis.com/oauth2/v1/userinfo",
                     headers={"Authorization": f"Bearer
{tokens['access_token']}"})
    if r.status_code != 200:
        tokens = refresh_access_token(tokens)
```

```
    return tokens["access_token"]

def load_signature():
    if os.path.exists(SIGNATURE_PATH):
        with open(SIGNATURE_PATH) as f:
            return f.read()
    return ""

def send_email(to, subject, html_body, dry_run=False):
    signature = load_signature()
    full_html = html_body + "\n<br>\n<div>--</div>\n" + signature
    msg = MIMEText(full_html, "html")
    msg["to"] = to
    msg["subject"] = subject
    raw = base64.urlsafe_b64encode(msg.as_bytes()).decode()
    if dry_run:
        print(f"[DRY RUN] Would send to: {to}")
        print(f"[DRY RUN] Subject: {subject}")
        return {"status": "dry_run"}
    token = get_valid_token()
    r = requests.post(
        "https://gmail.googleapis.com/gmail/v1/users/me/messages/send",
        headers={"Authorization": f"Bearer {token}", "Content-Type":
"application/json"},
        json={"raw": raw}
    )
    result = r.json()
    if r.status_code == 200:
        print(f"Email sent successfully. ID: {result.get('id')}")
    else:
        print(f"Failed to send: {r.status_code} - {r.text}")
    return result

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--to", required=True)
    parser.add_argument("--subject", required=True)
    parser.add_argument("--body-file", required=True)
    parser.add_argument("--dry-run", action="store_true")
    args = parser.parse_args()
    with open(args.body_file) as f:
        body = f.read()
    send_email(args.to, args.subject, body, args.dry_run)
```

check_replies.py — Inbox Monitor & Follow-up Manager

```
#!/usr/bin/env python3
"""Check inbox for replies to outreach emails and manage follow-ups.
Usage: python check_replies.py [--check-only] [--status]
"""
import argparse, json, os, requests
from datetime import datetime, timedelta

CREDS_PATH = "/path/to/your/gmail_credentials.json"
CLIENT_SECRET_PATH = "/path/to/your/client_secret.json"
CRM_PATH = "/path/to/your/crm_tracker.json"

def get_valid_token():
    with open(CREDS_PATH) as f:
        tokens = json.load(f)
    r = requests.get("https://www.googleapis.com/oauth2/v1/userinfo",
                     headers={"Authorization": f"Bearer
{tokens['access_token']}"})
    if r.status_code != 200:
        with open(CLIENT_SECRET_PATH) as f:
            creds = json.load(f)
        key = "installed" if "installed" in creds else "web"
        refresh = requests.post("https://oauth2.googleapis.com/token", data=
{
            "refresh_token": tokens["refresh_token"],
            "client_id": creds[key]["client_id"],
            "client_secret": creds[key]["client_secret"],
            "grant_type": "refresh_token"
        })
        if refresh.status_code == 200:
            tokens["access_token"] = refresh.json()["access_token"]
            with open(CREDS_PATH, "w") as f:
                json.dump(tokens, f, indent=2)
        else:
            raise RuntimeError(f"Token refresh failed: {refresh.text}")
    return tokens["access_token"]

def load_crm():
    with open(CRM_PATH) as f:
        return json.load(f)

def save_crm(data):
    with open(CRM_PATH, "w") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
```

```
def search_replies(token, contact_email):
    query = f"from:{contact_email}"
    r = requests.get(
        "https://gmail.googleapis.com/gmail/v1/users/me/messages",
        headers={"Authorization": f"Bearer {token}"},
        params={"q": query, "maxResults": 5}
    )
    if r.status_code == 200:
        return r.json().get("messages", [])
    return []

def get_message_snippet(token, msg_id):
    r = requests.get(
        f"https://gmail.googleapis.com/gmail/v1/users/me/messages/{msg_id}",
        headers={"Authorization": f"Bearer {token}"},
        params={"format": "metadata", "metadataHeaders": ["Subject", "Date",
"From"]}
    )
    if r.status_code == 200:
        data = r.json()
        snippet = data.get("snippet", "")
        date = ""
        for h in data.get("payload", {}).get("headers", []):
            if h["name"] == "Date":
                date = h["value"]
        return {"snippet": snippet, "date": date, "id": msg_id}
    return None

def check_all_replies():
    token = get_valid_token()
    crm = load_crm()
    today = datetime.now().strftime("%Y-%m-%d")
    results = {"replied": [], "no_reply": [], "due_follow_up": []}

    for lead in crm["leads"]:
        if lead["status"] == "closed":
            continue
        replies = search_replies(token, lead["contact_email"])
        if replies:
            for msg in replies:
                info = get_message_snippet(token, msg["id"])
                if info:
                    lead["replied"] = True
                    lead["reply_date"] = today
                    lead["reply_summary"] = info["snippet"][:200]
```

```
                    lead["status"] = "replied"
                    results["replied"].append({
                        "company": lead["company"],
                        "contact": lead["contact_name"],
                        "snippet": info["snippet"][:200],
                        "date": info["date"]
                    })
                    break
        else:
            if lead.get("next_follow_up") and lead["next_follow_up"] <=
today:
                if lead["follow_ups_sent"] < lead.get("max_follow_ups", 3):
                    results["due_follow_up"].append({
                        "company": lead["company"],
                        "contact": lead["contact_name"],
                        "email": lead["contact_email"],
                        "follow_ups_sent": lead["follow_ups_sent"]
                    })
            results["no_reply"].append({
                "company": lead["company"],
                "contact": lead["contact_name"],
                "days_waiting": (datetime.now() - datetime.strptime(
                    lead["first_sent"], "%Y-%m-%d")).days
            })

    save_crm(crm)
    return results

def print_status():
    crm = load_crm()
    print(f"=== SDR CRM Status ({datetime.now().strftime('%Y-%m-%d %H:%M')})
===")
    print(f"Total leads: {len(crm['leads'])}")
    for lead in crm["leads"]:
        status_icon = {"sent": "📤", "replied": "✅", "follow_up": "🔄",
"closed": "❌"}.get(lead["status"], "❓")
        print(f"{status_icon} {lead['company']} | {lead['contact_name']} |
{lead['status']}")
        print(f"   Email: {lead['contact_email']}")
        print(f"   Sent: {lead['first_sent']} | Follow-ups:
{lead['follow_ups_sent']}/{lead.get('max_follow_ups', 3)}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--check-only", action="store_true")
    parser.add_argument("--status", action="store_true")
```

```
    args = parser.parse_args()

    if args.status:
        print_status()
    else:
        results = check_all_replies()
        print(json.dumps(results, indent=2, ensure_ascii=False))
        if results["replied"]:
            print(f"\n🎉 {len(results['replied'])} lead(s) replied!")
        if results["due_follow_up"]:
            print(f"\n📬 {len(results['due_follow_up'])} lead(s) due for
follow-up")
```

Prompt Enhancer Skill (Bonus)

Used to optimize complex task prompts before execution. Based on patterns from Cursor, Claude Code, Devin AI, Amp, and Antigravity system prompts.

When to Activate

Apply when a task has 2+ of these signals:

- Multiple data sources or APIs involved
- Multi-step workflow (research → analyze → produce → verify)
- Deliverable with quality standards
- Integration with external systems
- User request is broad or underspecified

Enhancement Workflow

1. **Classify Complexity**: Simple / Medium / Complex
2. **Decompose into Phases**: Each with Goal, Inputs, Actions, Output, Gate
3. **Define Guardrails**: Source attribution, tone matching, error handling
4. **Set Output Spec**: Format, structure, quality bar, delivery method
5. **Build Verification Checklist**: Pre-delivery quality checks

Key Patterns

Pattern	Use Case
Parallel-then-Merge	Multiple sources → query in parallel → merge
Progressive Refinement	Broad → filter → deep analysis
Verify-Before-Commit	Generate → review → approve → execute
Fallback Chain	API → scraping → manual research
Context Anchoring	Re-read original request every 2-3 phases

Setup Requirements

To replicate this pipeline you need:

1. **Apollo.io account** (free tier works for company search)
2. **Google Cloud project** with Gmail API enabled + OAuth credentials
3. **Python packages**: `dnspython`, `requests`, `holehe`
4. **Crunchbase API** (optional, via RapidAPI)

Installation

```
pip install dnspython requests holehe
```

OAuth Setup

1. Create a project in Google Cloud Console
2. Enable Gmail API
3. Create OAuth 2.0 credentials (Desktop app)
4. Run the auth flow once to get refresh token
5. Save credentials JSON securely

Results

First day of operation:

- 4 leads identified, researched, and contacted
- 100% email verification rate (confidence 9-$^{10}/_{10}$)
- Average pipeline time: ~5 minutes per lead
- Total cost: $0 (all open source tools)

Built with Manus AI by Amir Baldiga