Trading Automation Apis Checklist

Signal Pilot Education Hub

🔌 Trading Automation & APIs Checklist

Lesson 37: Trading Automation APIs

This checklist guides you through connecting to broker APIs, building trading bots, and deploying automated strategies with proper error handling and monitoring.

📋 Phase 1: API Setup & Connection

Broker API Selection

- [ ] **Alpaca (Stocks & Crypto)** - Commission-free, clean API, paper trading built-in
- [ ] **Interactive Brokers (Multi-Asset)** - Stocks, options, futures, forex (professional-grade)

- [ ] **Binance (Crypto)** - Largest crypto exchange, 0.1% fees, 24/7 markets
- [ ] **TD Ameritrade / Schwab** - ThinkorSwim API (US stocks, options)

API Credentials & Security

- [ ] **Generate API keys** - API Key + Secret (never share or hardcode)
- [ ] **Store in environment variables** - Use `.env` file or system env vars (NOT in code)
- [ ] **Set IP whitelist (if available)** - Restrict API access to your IP only
- [ ] **Enable 2FA on broker account** - Add security layer beyond API keys
- [ ] **Use paper trading first** - Test with fake money (Alpaca, Binance testnet)

Test Connection

- [ ] **Connect to API** - Initialize connection with credentials
- [ ] **Fetch account info** - Verify buying power, equity, positions
- [ ] **Test data retrieval** - Pull latest price for SPY or BTC
- [ ] **Test order placement (paper)** - Place test order, verify it works
- [ ] **Check rate limits** - Most APIs limit requests (e.g., 200/min)

🎯 Phase 2: Building Trading Bot

Bot Architecture Design

- [ ] **Define strategy logic** - Janus sweep, breakout, mean reversion?
- [ ] **Data fetching module** - Pull real-time or historical data (REST or WebSocket)

- [ ] **Signal generation module** - Check entry conditions (if X and Y, then signal)
- [ ] **Risk management module** - Calculate position size, check portfolio heat
- [ ] **Order execution module** - Place bracket orders (entry + stop + target)
- [ ] **Monitoring & logging module** - Track performance, log errors

Core Bot Functions (Python Example)

```python
# 1. Connect to API
def connect_api():
    api_key = os.getenv('ALPACA_API_KEY')
    api_secret = os.getenv('ALPACA_API_SECRET')
    return tradeapi.REST(api_key, api_secret, base_url)

# 2. Fetch market data
def get_latest_price(symbol):
    bars = api.get_barset(symbol, 'minute', limit=50)
    return bars[symbol][-1].c

# 3. Check entry conditions
def check_sweep_setup(bars):
    swing_low = min([bar.l for bar in bars[-20:]])
    current_low = bars[-1].l
    current_price = bars[-1].c
    if current_low < swing_low * 0.997 and current_price > swing_low:
        return True  # Sweep confirmed
    return False

# 4. Calculate position size
def calculate_size(entry, stop, risk_pct=0.02):
    account = api.get_account()
    risk_amount = float(account.equity) * risk_pct
    stop_distance = abs(entry - stop)
    return int(risk_amount / stop_distance)

# 5. Place bracket order
def place_trade(symbol, size, entry, stop, target):
    order = api.submit_order(
        symbol=symbol,
        qty=size,
        side='buy',
        type='limit',
        limit_price=entry,
        time_in_force='gtc',
        order_class='bracket',
        stop_loss={'stop_price': stop},
        take_profit={'limit_price': target}
    )
    return order
```

Implement Core Features

- [ ] **Main loop** - Check for setups every 60 seconds (or real-time WebSocket)
- [ ] **Position tracking** - Track open positions, don't enter duplicate trades
- [ ] **Risk checks** - Max 3 positions, max 8% portfolio heat
- [ ] **Logging** - Log every action (entry, exit, error) to file
- [ ] **Error handling** - Wrap API calls in try/except (network can fail)

📊 Phase 3: Error Handling & Robustness

Common Errors to Handle

- [ ] **Connection timeout** - Retry 3 times with exponential backoff (1s, 2s, 4s)
- [ ] **Order rejected** - Check: insufficient funds, invalid price, market closed
- [ ] **Partial fills** - Handle: cancel remaining or market-fill
- [ ] **Rate limit exceeded** - Sleep, then retry (respect API limits)
- [ ] **Market closed** - Check market hours before placing orders

Retry Logic (Example)

```
def place_order_with_retry(api, **order_params):
    max_retries = 3
    for attempt in range(max_retries):
        try:
            order = api.submit_order(**order_params)
            return order
        except Exception as e:
            if attempt < max_retries - 1:
                wait = 2 ** attempt  # Exponential backoff
                logger.warning(f"Order failed: {e}. Retry in {wait}s...")
                time.sleep(wait)
            else:
                logger.error(f"Order failed after {max_retries} attempts: {e}")
                send_alert("Order Failure", str(e))
                raise
```

Pre-Trade Validation

- [ ] **Check account balance** - Sufficient buying power?
- [ ] **Verify market hours** - US stocks: 9:30 AM - 4 PM ET
- [ ] **Validate price** - Is limit price within 2% of current price? (sanity check)
- [ ] **Check existing positions** - Already in this symbol? (avoid duplicates)
- [ ] **Verify risk limits** - Portfolio heat < 8%?

Kill-Switch Implementation

- [ ] **Daily loss limit** - If account down > -3%, stop trading for the day
- [ ] **Consecutive loss limit** - 5 losses in a row → pause for 24 hours
- [ ] **Drawdown limit** - Max drawdown > -20% → email alert + stop trading
- [ ] **Error rate threshold** - > 10 API errors/hour → pause bot, investigate

---

📊 Phase 4: Monitoring & Alerts

Real-Time Monitoring Dashboard

- [ ] **Track current positions** - Symbol, size, entry, unrealized P&L
- [ ] **Monitor portfolio metrics** - Total P&L (day/week/month), portfolio heat
- [ ] **Check bot health** - Last heartbeat, API latency, error count
- [ ] **Display recent trades** - Last 10 trades with outcomes

Alert System Setup

```
import smtplib
from email.mime.text import MIMEText

def send_alert(subject, message):
    msg = MIMEText(message)
    msg['Subject'] = subject
    msg['From'] = 'bot@yourbot.com'
    msg['To'] = 'you@gmail.com'

    with smtplib.SMTP('smtp.gmail.com', 587) as server:
        server.starttls()
        server.login('bot@yourbot.com', os.getenv('EMAIL_PASSWORD'))
        server.send_message(msg)
```

Alert Triggers

- [ ] **Trade executed** - Telegram/Email: "Bought 100 SPY @ $520"
- [ ] **Stop hit** - "Stop loss triggered: -1R on SPY"
- [ ] **Target hit** - "Take profit hit: +3R on QQQ"
- [ ] **Error occurred** - "API error: Connection timeout (retrying...)"
- [ ] **Kill-switch activated** - "ALERT: Daily loss > -3%, trading halted"

Logging Best Practices

- [ ] **Log to file** - `bot.log` with timestamps
- [ ] **Log levels** - INFO (trades), WARNING (retries), ERROR (failures)
- [ ] **Log all API calls** - Request + response (debugging)
- [ ] **Rotate logs** - Daily rotation, keep last 30 days

📊 Phase 5: Deployment & Maintenance

Hosting Options

- [ ] **Local PC** - Cons: Power outages, internet fails, restarts (NOT recommended)
- [ ] **Cloud VPS** - AWS, DigitalOcean, Linode ($5-20/month, 99.9% uptime)
- [ ] **Raspberry Pi** - Low cost, runs 24/7, but limited power (OK for simple bots)

Deployment Checklist

- [ ] **Set up VPS** - Ubuntu 22.04, 2GB RAM, 1 CPU (sufficient for most bots)
- [ ] **Install dependencies** - Python, pip, libraries (alpaca-trade-api, etc.)
- [ ] **Upload bot code** - Git clone or SCP upload
- [ ] **Set environment variables** - API keys in `.env` file
- [ ] **Run as service** - Use `systemd` or `supervisor` (auto-restart on crash)
- [ ] **Test in paper mode** - Run for 1 week, verify stability

Auto-Restart Configuration

```
# systemd service file: /etc/systemd/system/trading-bot.service
[Unit]
Description=Trading Bot
After=network.target
```

```
[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/trading-bot
ExecStart=/usr/bin/python3 bot.py
Restart=always
RestartSec=30

[Install]
WantedBy=multi-user.target
```

Monthly Maintenance

- [ ] **Review performance** - Win rate, avg R, drawdown vs. backtest
- [ ] **Check error logs** - Recurring errors? Fix them
- [ ] **Update dependencies** - Pip upgrade alpaca-trade-api, etc.
- [ ] **Verify API keys** - Still valid? Expiring soon?
- [ ] **Regime check** - Is market regime suitable for this strategy?

💡 Pro Tips

Automation Mastery

- **Paper trade for 3-6 months** - Prove bot works before risking real capital
- **Start with 50% of intended size** - Scale up after 30+ successful live trades
- **Monitor daily for first month** - Catch bugs early before they compound
- **Keep it simple** - Fewer lines of code = fewer bugs

Common Mistakes to Avoid

- ❌ Hardcoding API keys in code (security risk)
- ❌ No error handling (bot crashes on first API timeout)
- ❌ No kill-switch (bot bleeds account during bad streak)
- ❌ Running on home PC (internet/power outages kill bot)
- ❌ Skipping paper trading (going live = expensive lesson)

Bot Optimization Tips

- **Use WebSockets for real-time data** - Faster than polling REST API every minute
- **Batch API calls** - Fetch multiple symbols at once (reduce requests)
- **Cache data** - Don't re-fetch same data multiple times
- **Asynchronous execution** - Use `asyncio` for parallel API calls

Production Best Practices

- **Version control** - Git repo for code (track changes)
- **Separate paper/live configs** - Different API keys, easy to switch
- **Backup trades to database** - SQLite or PostgreSQL (for analysis)
- **Monitor from mobile** - Install mobile SSH app (check bot on the go)

📚 Related Resources

- **Lesson 34:** System Development (design strategy before automating)
- **Lesson 35:** Machine Learning Trading (ML-powered bots)
- **Lesson 41:** Professional Infrastructure (cloud hosting, redundancy)

- **Recommended Tools:** Alpaca API, Python (alpaca-trade-api), DigitalOcean VPS

Version: 1.0
Last Updated: 2025-11-02
Difficulty: Advanced

Remember: Automation amplifies your edge—and your mistakes. Test rigorously. Deploy cautiously. Monitor continuously. A well-built bot trades perfectly. A broken bot destroys accounts.
