TickFlow Capital

Kill Switch Implementation Guide

Redis-Based Trading Halt System

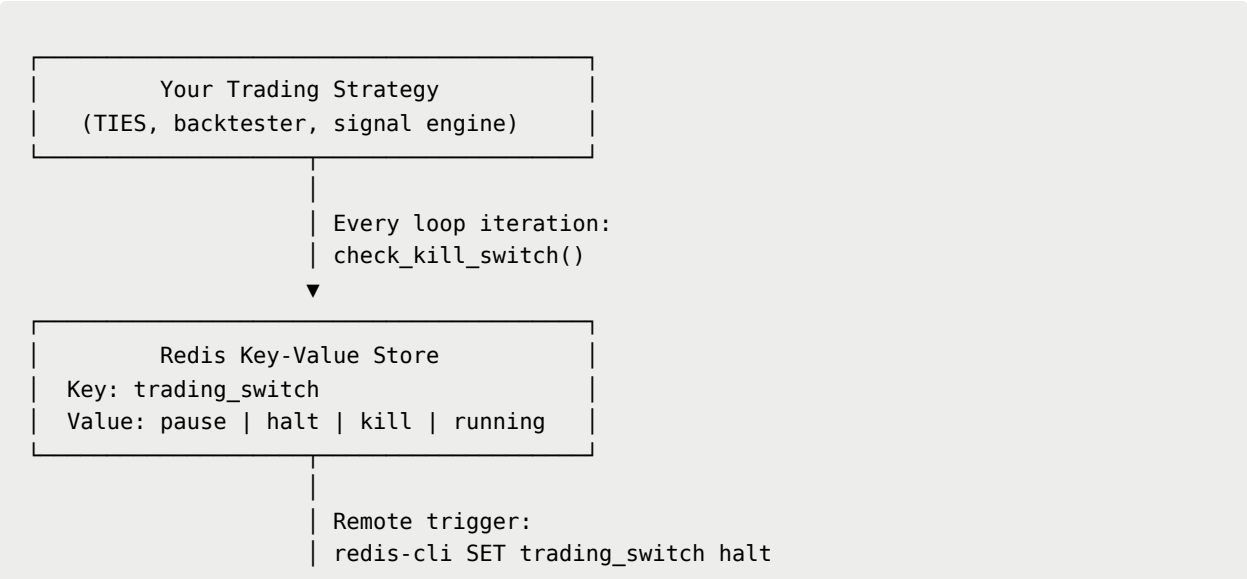
v1.0 – April 2026

Overview

A kill switch is a critical safety mechanism that allows immediate halting of trading operations. This guide covers the implementation of a three-tier Redis-based system:

- **PAUSE** — Stop new signal generation; keep positions open
- **HALT** — Stop new orders; retain positions for manual management
- **KILL** — Emergency liquidation; close all positions immediately

Architecture Overview



```
                    |
                    ▼
            Your VPS / Terminal
```

Part 1: Redis Installation & Configuration

Step 1: Install Redis

```
sudo apt update
sudo apt install redis-server -y
sudo systemctl enable redis-server
sudo systemctl start redis-server
```

Step 2: Verify Installation

```
redis-cli ping
# Expected output: PONG
```

Step 3: Secure Redis with Password

Edit `/etc/redis/redis.conf`:

```
sudo nano /etc/redis/redis.conf
```

Find and uncomment the `requirepass` line, then set a strong password:

```
requirepass YOUR_STRONG_PASSWORD_HERE
```

Example (DO NOT use this in production):

```
requirepass trading_switch_2026_secure
```

Step 4: Restrict Network Access

In the same file, ensure Redis only listens on localhost:

```
bind 127.0.0.1
```

For remote access (VPS scenario), bind to your VPS IP:

```
bind YOUR_VPS_IP
```

Step 5: Restart Redis

```
sudo systemctl restart redis-server
```

Step 6: Test Password Authentication

```
redis-cli -a YOUR_STRONG_PASSWORD ping
# Expected output: PONG
```

Part 2: Python Integration

Installation

```
pip install redis
```

Complete Python Code

Integrate this into your main trading loop:

```
import redis
import sys
import os
import time
from datetime import datetime
```

```
# Initialize Redis connection
r = redis.Redis(
    host='localhost',
    port=6379,
    password=os.getenv('REDIS_PASSWORD'),
    decode_responses=True,
    socket_connect_timeout=5
)

def check_kill_switch():
    """
    Check the current kill switch state and take appropriate
action.

    States:
    - 'running': normal operation
    - 'pause': skip new signals, keep positions open
    - 'halt': skip orders, keep positions open
    - 'kill': emergency liquidation, close all positions
    """
    try:
        state = r.get('trading_switch')

        if state is None:
            # Default to running if key doesn't exist
            r.set('trading_switch', 'running')
            return 'running'

        state = state.strip().lower()

        if state == 'pause':
            print(f"[{datetime.now().isoformat()}] ⚠  PAUSE:
No new trades")
            return 'pause'

        elif state == 'halt':
```

```
            print(f"[{datetime.now().isoformat()}] 🛑 HALT: C
losing positions")
            close_all_positions()
            return 'halt'

        elif state == 'kill':
            print(f"[{datetime.now().isoformat()}] 🔥 KILL: E
mergency liquidation")
            liquidate_all_positions(immediately=True)
            sys.exit(1)  # Exit the process

        else:
            return 'running'

    except redis.ConnectionError as e:
        print(f"❌ Redis connection error: {e}")
        # Fail-safe: consider it a halt if Redis is unreachab
le
        return 'halt'

    except Exception as e:
        print(f"❌ Unexpected error in check_kill_switch:
{e}")
        return 'halt'

def close_all_positions():
    """
    Gracefully close all open positions.
    Keep this separate from liquidate_all_positions for contr
ol.
    """
    try:
        # Example: iterate through MT5 positions
        # positions = mt5.positions_get()
        # for p in positions:
        #     close_position(p.ticket, p.symbol, p.volume)
```

```
        print("Closing all open positions...")
        # TODO: implement position closing logic
    except Exception as e:
        print(f"Error closing positions: {e}")

def liquidate_all_positions(immediately=False):
    """
    Emergency liquidation: close all positions at market.
    Set immediately=True to use market orders (no limit).
    """
    try:
        print("🔥 EMERGENCY LIQUIDATION IN PROGRESS")
        # Example: use MT5 API
        # positions = mt5.positions_get()
        # for p in positions:
        #     order_type = mt5.ORDER_TYPE_SELL if p.type == 0
else mt5.ORDER_TYPE_BUY
        #     request = {
        #         'action': mt5.TRADE_ACTION_DEAL,
        #         'symbol': p.symbol,
        #         'volume': p.volume,
        #         'type': order_type,
        #         'type_filling': mt5.ORDER_FILLING_IOC,  # I
mmediate-or-Cancel
        #     }
        #     result = mt5.order_send(request)
        #     print(f"Liquidated {p.symbol}: retcode {result.
retcode}")
        print("Emergency liquidation complete")
    except Exception as e:
        print(f"Error during emergency liquidation: {e}")

# ──────────────────────────────────────────────────────────
──────
# Main Trading Loop Integration
# ──────────────────────────────────────────────────────────
```

```
_______


def main_trading_loop():
    """Your main trading loop."""
    loop_count = 0

    while True:
        try:
            loop_count += 1

            # CHECK KILL SWITCH AT START OF EVERY ITERATION
            status = check_kill_switch()

            if status == 'pause':
                # Skip signal generation but keep positions
                time.sleep(1)
                continue

            if status == 'halt':
                # Stop new orders but don't close positions
                time.sleep(1)
                continue

            # Normal operation (status == 'running')

            # Your signal generation logic
            # signals = signal_engine.scan(tick, features)

            # Your risk evaluation logic
            # decision = risk_engine.evaluate(signal)

            # Your execution logic
            # result = execution_bridge.execute(signal, decis
ion)

            time.sleep(1)
```

```
            if loop_count % 60 == 0:
                print(f"[{datetime.now().isoformat()}] Loop i
teration {loop_count}")

        except KeyboardInterrupt:
            print("\nShutdown signal received")
            sys.exit(0)

        except Exception as e:
            print(f"Error in main loop: {e}")
            time.sleep(1)

if __name__ == "__main__":
    print("Starting trading system with Redis kill switc
h...")
    main_trading_loop()
```

Environment Setup

Create a `.env` file in your project directory:

```
REDIS_PASSWORD=your_strong_password_here
REDIS_HOST=localhost
REDIS_PORT=6379
```

Load it in your startup script:

```
export $(cat .env | xargs)
python main.py
```

Part 3: Remote Trigger Commands

From Your Local Machine

Trigger the kill switch remotely using `redis-cli`:

```
# PAUSE – Stop new signals
redis-cli -h YOUR_VPS_IP -a PASSWORD SET trading_switch pause

# HALT – Stop orders
redis-cli -h YOUR_VPS_IP -a PASSWORD SET trading_switch halt

# KILL – Emergency liquidation
redis-cli -h YOUR_VPS_IP -a PASSWORD SET trading_switch kill

# RESET – Return to normal operation
redis-cli -h YOUR_VPS_IP -a PASSWORD SET trading_switch runni
ng

# CHECK – View current state
redis-cli -h YOUR_VPS_IP -a PASSWORD GET trading_switch
```

Python Remote Trigger Script

```
import redis

def trigger_kill_switch(vps_ip, password, action):
    """
    Remotely trigger the kill switch.
    action: 'pause', 'halt', 'kill', or 'running'
    """
    r = redis.Redis(host=vps_ip, port=6379, password=passwor
d, decode_responses=True)
    r.set('trading_switch', action)
    print(f"Kill switch set to: {action}")
    print(f"Current state: {r.get('trading_switch')}")
```

```
# Example usage:
trigger_kill_switch('your.vps.ip', 'PASSWORD', 'halt')
```

Part 4: Testing Checklist

Before deploying to live trading, verify:

- ☐ Redis service is running and auto-starts on reboot
- ☐ Password authentication works from your VPS
- ☐ Password authentication works from your local machine
- ☐ PAUSE state prevents new signals from being generated
- ☐ HALT state prevents new orders from being sent
- ☐ KILL state closes all positions without error
- ☐ State transitions are logged with timestamps
- ☐ Logs record the original state and the new state
- ☐ Emergency recovery procedure is documented
- ☐ All team members have the Redis password securely stored

Emergency Procedures

If Kill Switch Fails

1. **Immediately close all positions manually in MT5 terminal**
2. Check Redis connectivity:

   ```
   redis-cli -a PASSWORD ping
   ```

3. Manually verify trading loop has stopped:

   ```
   ps aux | grep python
   ```

4. If needed, force-kill the process:

```
pkill -9 python
```

If Redis Service Crashes

1. Restart Redis:

```
sudo systemctl restart redis-server
```

2. Verify it's running:

```
sudo systemctl status redis-server
```

3. Set trading_switch back to 'running' when ready:

```
redis-cli -a PASSWORD SET trading_switch running
```

Monitoring

Monitor kill switch activity with structured logging:

```
import logging

logger = logging.getLogger('kill_switch')
handler = logging.FileHandler('kill_switch.log')
formatter = logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

def check_kill_switch():
    # ... existing code ...
    logger.info(f"Kill switch state: {state}")
```

TickFlow Capital

Website: tickflowcapital.com

Version: v1.0 – April 2026

© 2026 Tickflow Capital Limited. All rights reserved.

Disclaimer: This guide is provided for educational purposes. Always test on a demo account first. The kill switch is a safety mechanism but is not a substitute for careful risk management.