```python
#!/usr/bin/env python3
"""
MonsterDog Benchmark Automation System
A realistic implementation of automated benchmark submission and tracking
"""

import json
import time
import os
import hashlib
import qrcode
import requests
from datetime import datetime, timedelta
from pathlib import Path
from typing import Dict, List, Optional
import logging
from dataclasses import dataclass, asdict

# Configuration
@dataclass
class BenchmarkConfig:
    name: str
    endpoint: str
    auth_token: Optional[str] = None
    enabled: bool = True

@dataclass
class MetricResult:
    benchmark: str
    score: float
    timestamp: str
    metadata: Dict = None

class MonsterDogDaemon:
    def __init__(self, config_path: str = "config.json"):
        self.config_path = config_path
        self.log_path = Path("logs/benchmark_metrics.ndjson")
        self.qr_output = Path("output/qr_codes")
        self.webhook_url = None

        # Setup logging
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
            handlers=[
                logging.FileHandler('logs/daemon.log'),
                logging.StreamHandler()
            ]
        )
        self.logger = logging.getLogger("MonsterDog")
        # Create directories
        self.log_path.parent.mkdir(exist_ok=True)
        self.qr_output.mkdir(parents=True, exist_ok=True)

        # Load configuration
        self.benchmarks = self._load_config()

    def _load_config(self) -> List[BenchmarkConfig]:
        """Load benchmark configuration"""
        default_config = {
            "benchmarks": [
                {"name": "MMLU", "endpoint": "https://api.example.com/mmlu", "enabled": True},
                {"name": "ARC-Challenge", "endpoint": "https://api.example.com/arc",
"enabled": True},
                {"name": "GSM8K", "endpoint": "https://api.example.com/gsm8k", "enabled":
True},
                {"name": "HumanEval", "endpoint": "https://api.example.com/humaneval",
"enabled": True}
            ],
            "webhook_url": "https://discord.com/api/webhooks/YOUR_WEBHOOK_HERE",
            "submission_interval": 3600  # 1 hour
        }

        if not os.path.exists(self.config_path):
            with open(self.config_path, 'w') as f:
                json.dump(default_config, f, indent=2)
            self.logger.info(f"Created default config at {self.config_path}")

        with open(self.config_path, 'r') as f:
            config = json.load(f)

        self.webhook_url = config.get("webhook_url")
        self.submission_interval = config.get("submission_interval", 3600)

        return [BenchmarkConfig(**b) for b in config["benchmarks"]]

    def simulate_benchmark_run(self, benchmark_name: str) -> MetricResult:
        """Simulate running a benchmark (replace with actual benchmark code)"""
        import random

        # Simulate realistic scores for different benchmarks
        score_ranges = {
            "MMLU": (0.25, 0.95),
            "ARC-Challenge": (0.20, 0.85),
            "GSM8K": (0.10, 0.90),
            "HumanEval": (0.15, 0.80)
        }

        min_score, max_score = score_ranges.get(benchmark_name, (0.0, 1.0))
        score = random.uniform(min_score, max_score)

        return MetricResult(
            benchmark=benchmark_name,
            score=round(score, 4),
            timestamp=datetime.utcnow().isoformat(),
            metadata={
                "model_version": "MonsterDog-v1.0",
                "temperature": 0.1,
                "max_tokens": 2048,
                "run_id": hashlib.md5(f"{benchmark_name}{time.time()}".encode()).hexdigest()
[:8]
            }
        )

    def log_metric(self, result: MetricResult):
        """Log benchmark result to NDJSON file"""
        with open(self.log_path, 'a') as f:
            json.dump(asdict(result), f)
            f.write('\n')

        self.logger.info(f"Logged {result.benchmark}: {result.score}")

    def generate_submission_payload(self, results: List[MetricResult]) -> Dict:
        """Generate submission payload"""
        timestamp = datetime.utcnow().isoformat()

        payload = {
            "entity_id": "MONSTERDOG_ENTITY72K",
            "timestamp": timestamp,
            "results": [asdict(r) for r in results],
            "signature": self._generate_signature(results),
            "version": "1.0.0"
        }

        return payload

    def _generate_signature(self, results: List[MetricResult]) -> str:
        """Generate cryptographic signature for results"""
        data = json.dumps([asdict(r) for r in results], sort_keys=True)
        return hashlib.sha256(data.encode()).hexdigest()

    def create_qr_code(self, payload: Dict) -> str:
        """Generate QR code for submission"""
        qr_data = {
            "entity_id": payload["entity_id"],
            "timestamp": payload["timestamp"],
            "signature": payload["signature"]
        }

        filename = f"submission_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
        filepath = self.qr_output / filename

        qr = qrcode.QRCode(version=1, box_size=10, border=5)
        qr.add_data(json.dumps(qr_data))
        qr.make(fit=True)
        img = qr.make_image(fill_color="black", back_color="white")
        img.save(filepath)

        self.logger.info(f"QR code generated: {filepath}")
        return str(filepath)

    def submit_to_benchmark(self, benchmark: BenchmarkConfig, payload: Dict) -> bool:
        """Submit results to benchmark platform"""
        try:
            headers = {"Content-Type": "application/json"}
            if benchmark.auth_token:
                headers["Authorization"] = f"Bearer {benchmark.auth_token}"

            # For demonstration, we'll just log the submission
            # In reality, you'd make an actual HTTP request
            self.logger.info(f"Submitting to {benchmark.name}: {benchmark.endpoint}")

            # Uncomment for actual submission:
            # response = requests.post(benchmark.endpoint, json=payload, headers=headers)
            # return response.status_code == 200

            return True  # Simulate success

        except Exception as e:
            self.logger.error(f"Submission to {benchmark.name} failed: {e}")
            return False

    def send_webhook_notification(self, results: List[MetricResult], qr_path: str):
        """Send Discord/Slack webhook notification"""
        if not self.webhook_url:
            return

        scores_summary = ", ".join([f"{r.benchmark}: {r.score:.3f}" for r in results])

        webhook_data = {
            "content": f"🤖 **MonsterDog Benchmark Update**\n```\n{scores_summary}\n```\nQR:
{qr_path}",
            "username": "MonsterDog Daemon"
        }

        try:
            # requests.post(self.webhook_url, json=webhook_data)
            self.logger.info("Webhook notification sent")
        except Exception as e:
            self.logger.error(f"Webhook failed: {e}")

    def run_benchmark_cycle(self):
        """Run one complete benchmark cycle"""
        self.logger.info("🚀 Starting benchmark cycle...")

        results = []
        for benchmark in self.benchmarks:
            if not benchmark.enabled:
```

```
                continue

            self.logger.info(f"Running {benchmark.name}...")
            result = self.simulate_benchmark_run(benchmark.name)
            self.log_metric(result)
            results.append(result)

        if results:
            # Generate submission payload
            payload = self.generate_submission_payload(results)

            # Create QR code
            qr_path = self.create_qr_code(payload)

            # Submit to enabled benchmarks
            for benchmark in self.benchmarks:
                if benchmark.enabled:
                    self.submit_to_benchmark(benchmark, payload)

            # Send notifications
            self.send_webhook_notification(results, qr_path)

            self.logger.info(f"✅ Cycle complete. {len(results)} benchmarks processed.")
        else:
            self.logger.warning("No benchmarks enabled or results generated.")

    def run_daemon(self):
        """Main daemon loop"""
        self.logger.info("🌀 MonsterDog Benchmark Daemon starting...")
        self.logger.info(f"Submission interval: {self.submission_interval} seconds")

        try:
            while True:
                self.run_benchmark_cycle()
                self.logger.info(f"💤 Sleeping for {self.submission_interval} seconds...")
                time.sleep(self.submission_interval)

        except KeyboardInterrupt:
            self.logger.info("🛑 Daemon stopped by user")
        except Exception as e:
            self.logger.error(f"💥 Daemon crashed: {e}")
            raise

# CLI Interface
def main():
    import argparse

    parser = argparse.ArgumentParser(description="MonsterDog Benchmark Automation System")
    parser.add_argument("--config", default="config.json", help="Configuration file path")
    parser.add_argument("--single-run", action="store_true", help="Run once instead of daemon
mode")
    parser.add_argument("--setup", action="store_true", help="Setup initial configuration")
    args = parser.parse_args()

    daemon = MonsterDogDaemon(args.config)

    if args.setup:
        print("✅ Configuration file created. Edit config.json to customize.")
        return

    if args.single_run:
        daemon.run_benchmark_cycle()
    else:
        daemon.run_daemon()

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