PC MetricsX: Real-Time Gaming PC Monitoring

Live thermal telemetry and AI-driven fan-curve optimization for gaming rigs.

Instructor: Guy Tel-Zur
Student: Elad Lavi | IoT Project

Objective & Motivation

AI-Driven Cooling

Cloud AI automatically creates optimal fan curves for peak performance.

Preventative Alerts

Real-time temperature warnings before hardware damage occurs.

Data Aggregation

Centralized telemetry from multiple gaming systems for better insights.

The Challenge

The Problem

- Systems reach 80°C+ under heavy gaming loads
- Manual fan tuning is inefficient, tedious, and endless (changing seasons, thermals degradation)
- No historical data tracking
- Difficulty of balancing noise level and good thermals

1

Sustained High Temps

Gaming PCs experience dangerous and prolonged high temperature during heavy sessions which degrade components overtime.

2

Cooling Issues

Users have suboptimal fan curves, risking performance throttling.

3

Noise Pollution

PC gamers report excessive fan noise distracting from their experience.

Solution Overview

Edge Collection

Python simulator emits JSON readings every second from PC sensors.

Cloud Ingest

AWS IoT Core routes data to Timestream database and Grafana dashboards.

Smart Alerts

CloudWatch & SNS triggers notifications when temperatures exceed thresholds.

AI Suggestions

Claude model analyzes patterns and recommends optimal fan curves.

Architecture Diagram

PC Simulator, Local Dashboard

Generates temperature and fan speed telemetry.

AWS IoT Core

Receives MQTT messages on gamingPC/telemetry topic.

Timestream DB & S3

Stores time-series metrics with device_id dimension.

Historical data archived to S3.

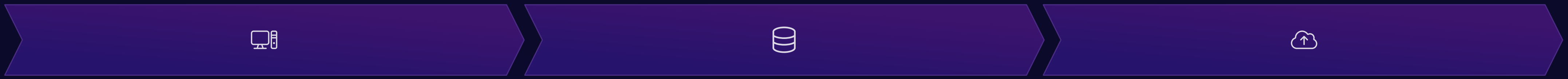
Grafana + Alerts (SNS, CloudWatch)

Visualizes data and sends SMS/email notifications.

Anthropic AI

Analyzes patterns from various PCs and recommends optimal fan curves.

Data Flow & Ingestion



PC Sensor Simulation

Python script generates realistic temperature fan speed, gaming intensity readings every second.

Local Dashboard

A Flask dashboard visualizes live sensor data from the simulator.

MQTT Publication

Device certificates authenticate secure connection to AWS IoT Core endpoint.

PC Metrics Simulation

```
# In main() loop
while True:
    pc_simulator.update_gaming_session()
    pc_simulator.simulate_temperature_changes()
    pc_simulator.calculate_fan_speeds()

    sensor_data = pc_simulator.get_sensor_data()
    # ... print to console, then:
    if aws_connected:
        aws_publisher.publish_data(sensor_data)
    time.sleep(config.SENSOR_UPDATE_INTERVAL)
```

```
class PCComponentSimulator:
    def update_gaming_session(self):
        # decides when to start/stop gaming and adjusts
self.gaming_intensity

    def simulate_temperature_changes(self):
        # updates self.cpu_temp, self.gpu_temp, etc. based
on self.gaming_intensity

    def calculate_fan_speeds(self):
        # sets self.cpu_fan_rpm, self.gpu_fan_rpm,
self.case_fan_rpm via fan curves

    def get_sensor_data(self):
        return {
            'timestamp': int(time.time()),
            'cpu_temp': round(self.cpu_temp, 1),
            'gpu_temp': round(self.gpu_temp, 1),
            # ... other fields ...
        }
```

Flask Local Dashboard

```
# background thread
def background_data_monitor():
    while True:
        load_sensor_data()
        check_aws_connection()
        if current_sensor_data:
            socketio.emit('sensor_update', {
                'data': current_sensor_data,
                'aws_status': aws_connection_status,
                'component_models': COMPONENT_MODELS
            })
        time.sleep(2)

# Flask routes
@app.route('/')
def dashboard():
    return render_template('dashboard.html',
component_models=COMPONENT_MODELS)

@app.route('/api/current_data')
def api_current_data():
    return jsonify({
        'current_data': current_sensor_data,
        'aws_connected': aws_connection_status,
        'component_models': COMPONENT_MODELS
    })
```

AWS IoT Core Connection

```
# In main()
aws_publisher = AWSIoTPublisher()
if aws_sdk_available:
    aws_connected = aws_publisher.connect()
    if aws_connected:
        print("SUCCESS: AWS IoT integration active")
```

```
class AWSIoTPublisher:
    def __init__(self):
        # ...
        self.mqtt_client = AWSIoTMQTTClient(self.client_id)

self.mqtt_client.configureEndpoint(config.AWS_IOT_ENDP
OINT, config.MQTT_PORT)
        self.mqtt_client.configureCredentials(
            config.ROOT_CA_PATH,
            config.PRIVATE_KEY_PATH,
            config.CERTIFICATE_PATH
        )
        # ... configure timeouts, queueing, etc.

    def connect(self):
        print("INFO: Connecting to AWS IoT Core...")
        self.mqtt_client.connect()
        self.connected = True
        print("SUCCESS: Successfully connected to AWS IoT
Core!")
        return True

    def publish_data(self, sensor_data):
        json_payload = json.dumps(sensor_data, indent=2)
        return self.mqtt_client.publish(config.MQTT_TOPIC,
json_payload, 1)
```

Data Flow & Ingestion

MQTT Topic Structure

```
gamingPC/telemetry/{device_id}
```

IoT Rule Query

```
SELECT * FROM 'gamingPC/telemetry'
```

JSON Payload Example

```
{
    "timestamp": 1750277013,
    "device_id": "GamingPC4",
    "cpu_temp": 41.2,
    "gpu_temp": 36.7,
    "ssd_temp": 35.3,
    "motherboard_temp": 32.9,
    "cpu_fan_rpm": 1165,
    "gpu_fan_rpm": 958,
    "case_fan_rpm": 747,
    "gaming_session": false,
    "gaming_intensity": 0.0
}
```

Grafana Dashboard

PC Components Metrics

Real time temp curves with threshold indicators.

Fan Response

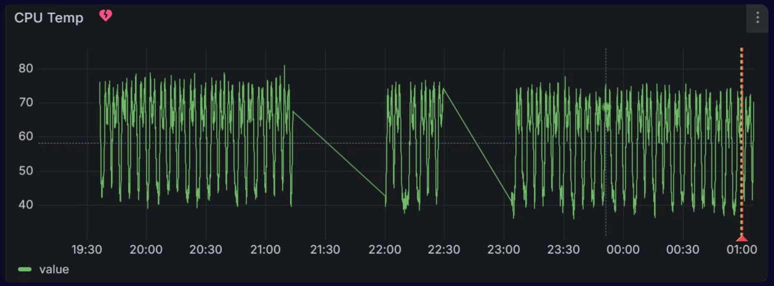
RPM curves showing cooling system reaction to load.

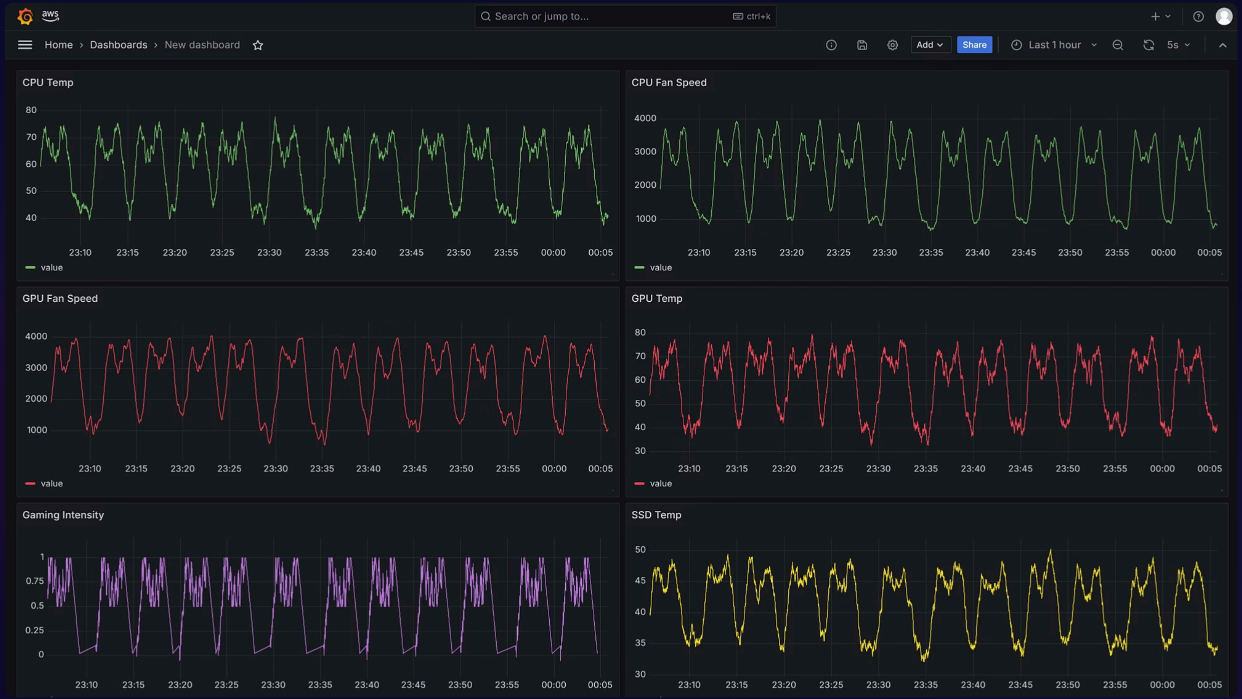
Alert Panel

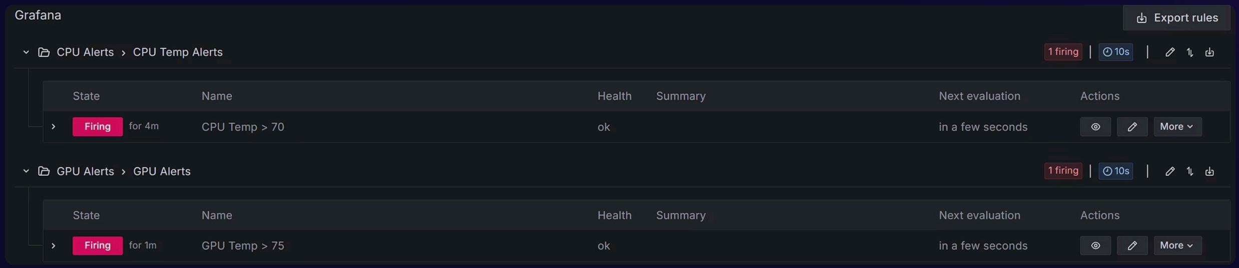
Active warnings of high temps displayed prominently.

Gaming Intensity

Shows real time indicator of gaming intensity in relation to load






Grafana

Export rules

CPU Alerts › CPU Temp Alerts — 1 firing | 10s

State	Name	Health	Summary	Next evaluation	Actions
Firing for 4m	CPU Temp > 70	ok		in a few seconds	More

GPU Alerts › GPU Alerts — 1 firing | 10s

State	Name	Health	Summary	Next evaluation	Actions
Firing for 1m	GPU Temp > 75	ok		in a few seconds	More

Local Dashboard Interface

Real-Time Visualization

Line graphs display temperature curves and fan RPM data with customizable time ranges. Cloud connection indicator shows AWS sync status.

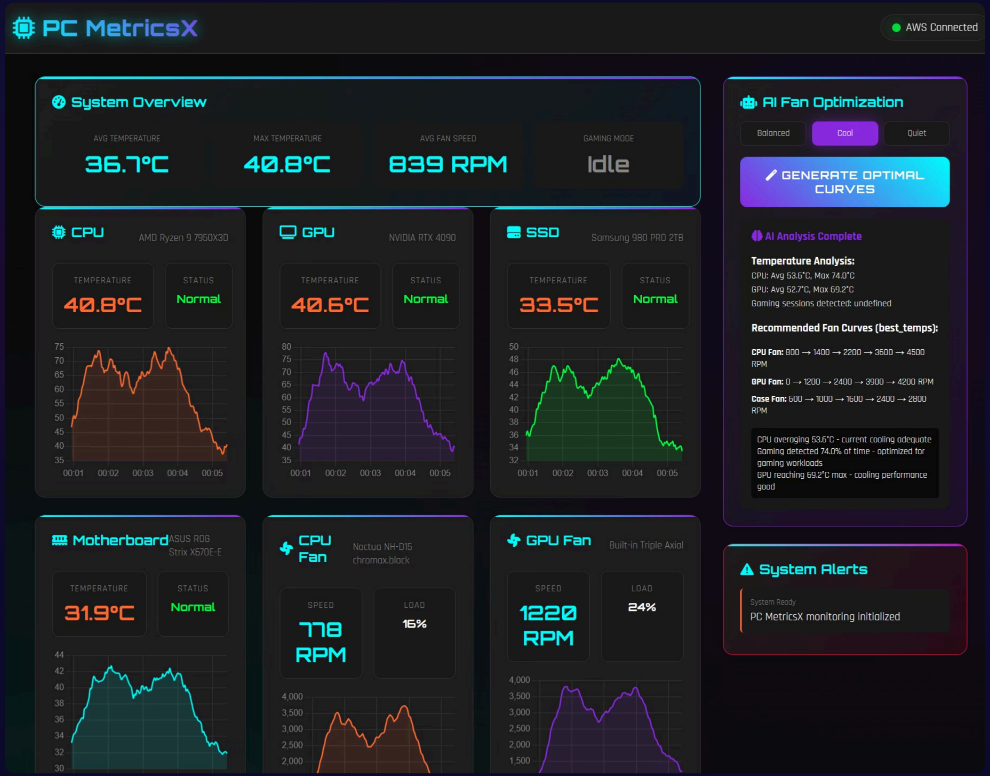
AI Recommendations

Smart suggestions optimize fan curves based on user preference for noise level or cooling performance.

Alert System

Visual warnings appear when temperatures approach critical thresholds.

Technologies: Python, Flask, AWS SDK, HTML, JavaScript



PC MetricsX

AWS Connected

System Overview

AVG TEMPERATURE 36.7°C

MAX TEMPERATURE 40.8°C

AVG FAN SPEED 839 RPM

GAMING MODE Idle

CPU AMD Ryzen 9 7950X3D

TEMPERATURE 40.8°C

STATUS Normal

GPU NVIDIA RTX 4090

TEMPERATURE 40.6°C

STATUS Normal

SSD Samsung 980 PRO 2TB

TEMPERATURE 33.5°C

STATUS Normal

Motherboard ASUS ROG Strix X670E-E

TEMPERATURE 31.9°C

STATUS Normal

CPU Fan Noctua NH-D15 chromax.black

SPEED 778 RPM

LOAD 16%

GPU Fan Built-in Triple Axial

SPEED 1220 RPM

LOAD 24%

AI Fan Optimization

AI Analysis Complete

Temperature Analysis:
CPU: Avg 53.6°C, Max 74.0°C
GPU: Avg 52.7°C, Max 69.2°C
Gaming sessions detected: undefined

Recommended Fan Curves (best_temps):

CPU Fan: 800 → 1400 → 2200 → 3600 → 4500 RPM

GPU Fan: 0 → 1200 → 2400 → 3900 → 4200 RPM

Case Fan: 600 → 1000 → 1600 → 2400 → 2800 RPM

CPU averaging 53.6°C - current cooling adequate
Gaming detected 74.0% of time - optimized for gaming workloads
GPU reaching 69.2°C max - cooling performance good

System Alerts

System Ready
PC MetricsX monitoring initialized

Security Architecture

Secure Connections

TLS-encrypted MQTT transmissions protect telemetry data in transit.

HTTPS dashboards ensure secure visualization access from any device.

Access Controls

Least-privilege IAM roles limit service permissions to absolute minimum.

Fine-grained policies restrict IoT, Timestream, and Grafana resources.

Notification Security

SNS topics protected by resource policies prevent unauthorized alerts.

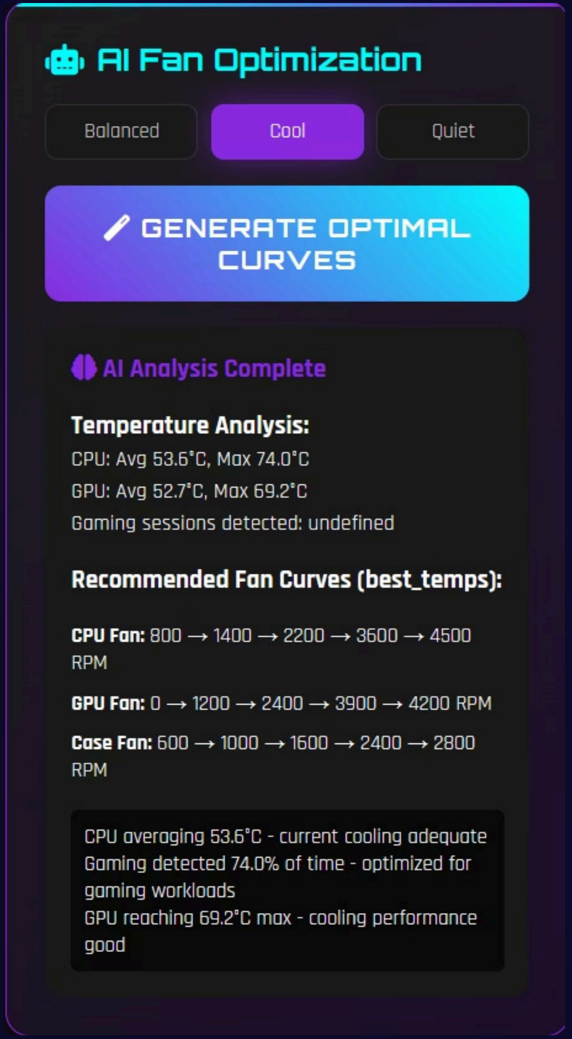
End-to-end encryption keeps notifications confidential across delivery channels.

Alerts & AI Recommendations via SNS and CloudWatch

Alert Triggers

CPU Temperature	> 70°C
GPU Temperature	> 75°C
SSD Temperature	> 75°C
Motherboard	> 80°C

AI Recommendation Examples



AI Analysis Complete

Temperature Analysis:

CPU: Avg 53.6°C, Max 74.0°C

GPU: Avg 52.7°C, Max 69.2°C

Gaming sessions detected: undefined

Recommended Fan Curves (best_temps):

CPU Fan: 800 → 1400 → 2200 → 3600 → 4500 RPM

GPU Fan: 0 → 1200 → 2400 → 3900 → 4200 RPM

Case Fan: 600 → 1000 → 1600 → 2400 → 2800 RPM

CPU averaging 53.6°C - current cooling adequate
Gaming detected 74.0% of time - optimized for gaming workloads
GPU reaching 69.2°C max - cooling performance good

Summary of Capabilities

Real-Time Monitoring

Continuous telemetry ingestion from PC components.

Intelligent Optimization

AI-driven fan curve generation adapts to usage patterns and thermal conditions.

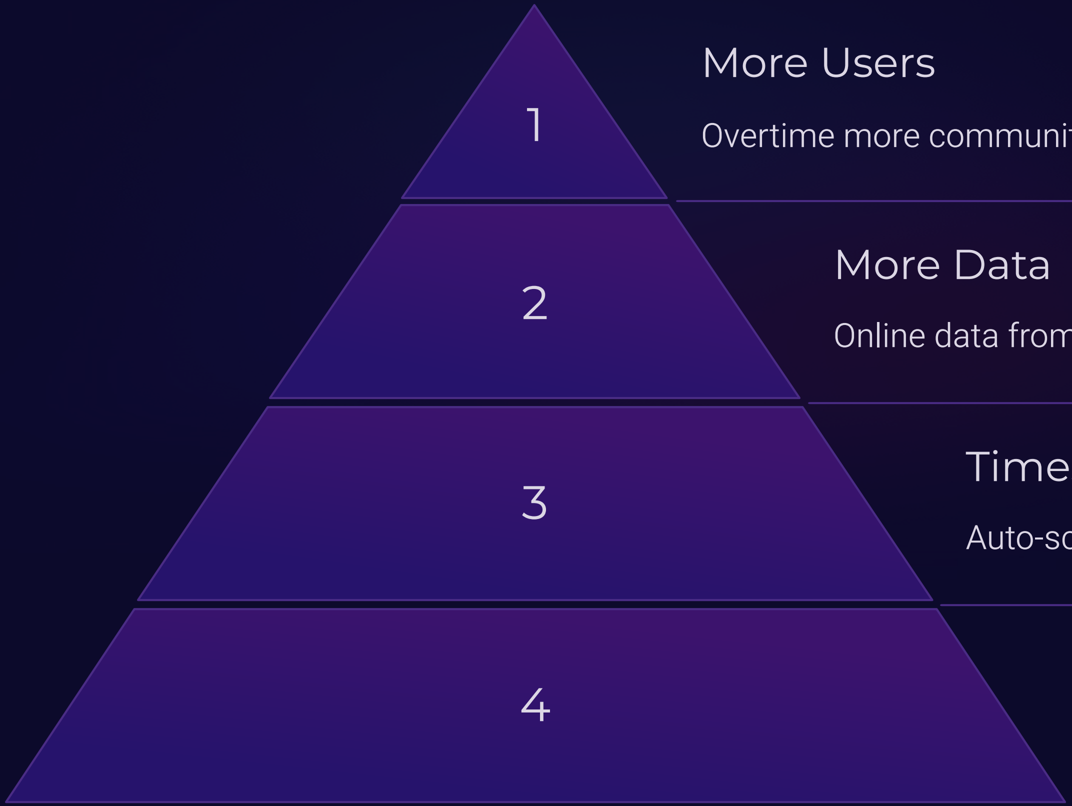
Proactive Protection

Automatic alerts prevent hardware degradation and damage from sustained high temps.

Serverless Architecture

Secure AWS infrastructure scales instantly with zero maintenance overhead.

Future Scalability



More Users

Overtime more community rigs

More Data

Online data from users of optimized fan curves and temp limits per PC component

Timestream

Auto-scales to millions of records/sec

SNS

Scales notifications to thousands of subscribers

Future Expansion

1. ## Community Onboarding

   Begin recruiting users to expand dataset and therefore improve AI suggestions.

2. ## API Development

   Create interfaces for third-party cooling software integration.

3. ## Mobile App

   Develop companion app for remote monitoring and alerts.

Thank You for Listening :)