Windows Server Monitoring System - Technical Deep Dive

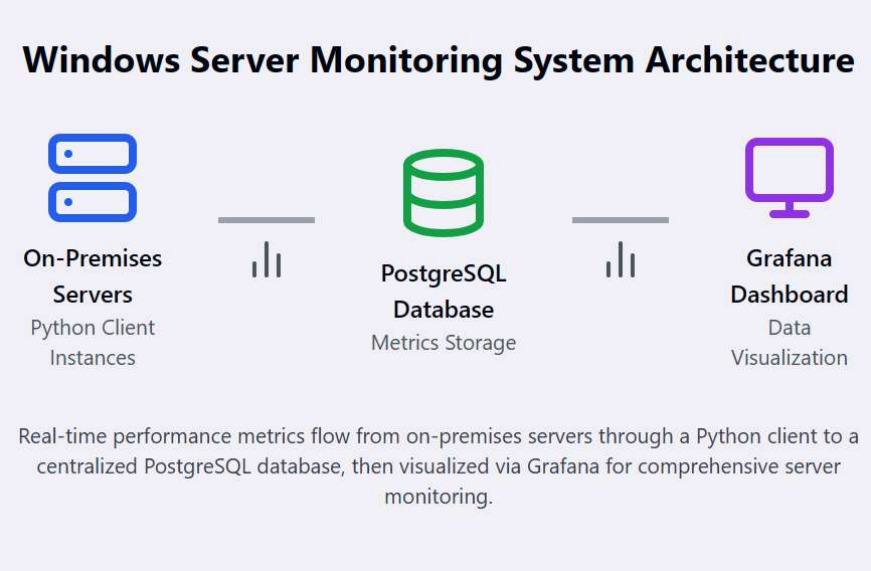
Project Overview

The Windows Server Monitoring System is a sophisticated, scalable solution for real-time performance tracking and analysis of on-premises servers using Python, PostgreSQL, and Grafana.

Author : Shriniwas Kulkarni

- **PCCOE 2026 BTech CSE(AIML)**
- **Email:** kshriniwas180205@gmail.com
- **Phone:** +91 [8999883480]
- **GitHub:** github.com/Shriniwas18K

Technical Architecture



System Components

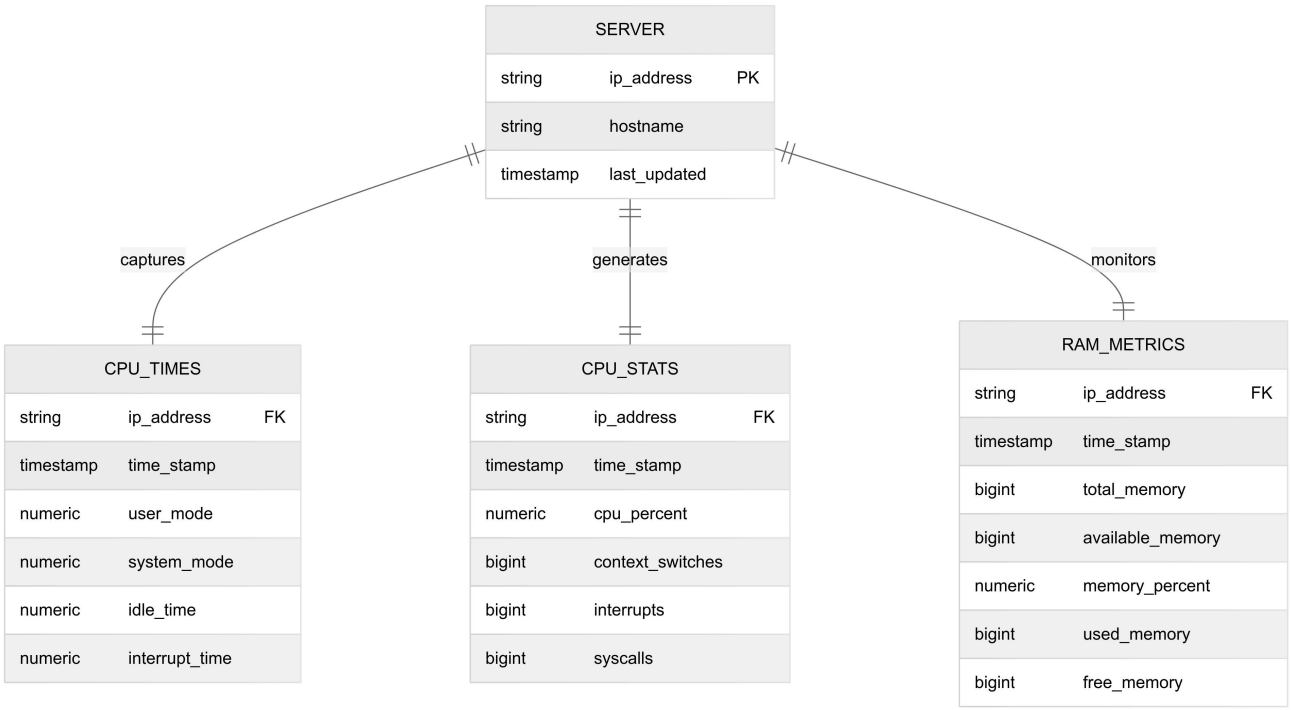
1. **Client-Side Monitoring**
    - **Technology**: Python with `psutil` library
    - **Key Functionality**:

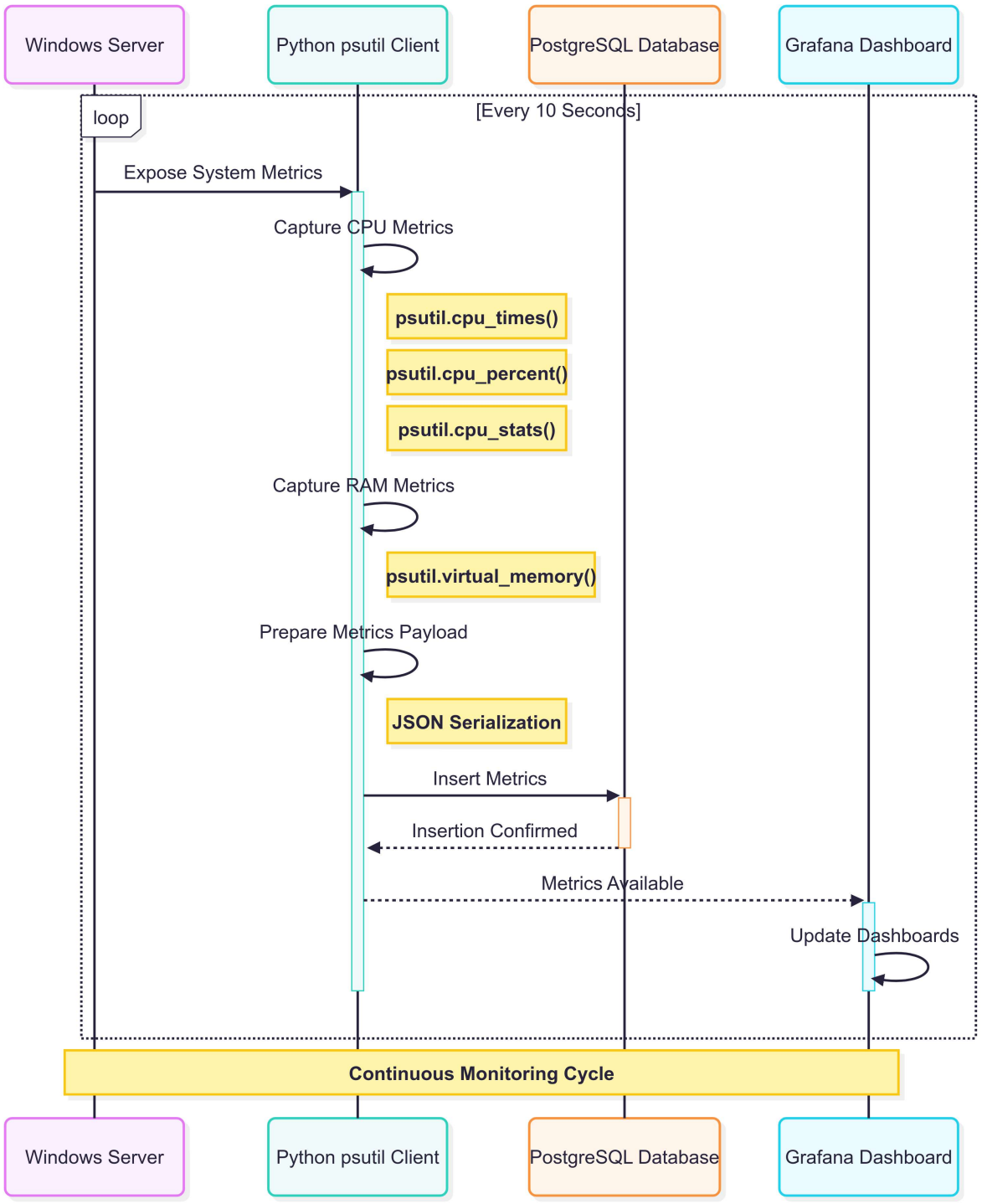
        - Real-time system metrics collection
        - Automated data aggregation
        - Periodic database transmission

2. **Database Storage**
    - **Technology**: PostgreSQL
    - **Data Schema**:
        - PHYSICALCPUTIMES
        - PHYSICALCPUSTATS
        - RAM

3. **Visualization**
    - **Technology**: Grafana
    - **Capability**: Interactive dashboard creation

Technical Deep Dive


SERVER
string ip_address PK
string hostname
timestamp last_updated
captures
generates
monitors
CPU_TIMES
string ip_address FK
timestamp time_stamp
numeric user_mode
numeric system_mode
numeric idle_time
numeric interrupt_time
CPU_STATS
string ip_address FK
timestamp time_stamp
numeric cpu_percent
bigint context_switches
bigint interrupts
bigint syscalls
RAM_METRICS
string ip_address FK
timestamp time_stamp
bigint total_memory
bigint available_memory
numeric memory_percent
bigint used_memory
bigint free_memory



Metrics Collection Mechanism

```
def main():
    while True:
        # CPU Metrics Capture
        req['physicalCPU']['cpuTimes'] = psutil.cpu_times()._asdict()
        req['physicalCPU']['cpuPercent'] = psutil.cpu_percent(interval=0.1)
        req['physicalCPU']['cpuStats'] = {
            field: getattr(psutil.cpu_stats(), field)
```

```
    for field in psutil.cpu_stats()._fields
}

# RAM Metrics Capture
req['RAM']['virtualMemory'] = {
    field: getattr(psutil.virtual_memory(), field)
    for field in psutil.virtual_memory()._fields
}

# Database Insertion
cur.execute('''
INSERT INTO PHYSICALCPUTIMES VALUES (
    ip, timestamp, user_mode, system_mode, idle, interrupt, dpc
)''')

time.sleep(10)  # 10-second interval between captures
```

Key Technical Highlights

- **Continuous Monitoring**: 24/7 metric collection
- **Low Overhead**: Lightweight Python script
- **Flexible Architecture**: Easily extensible to multiple servers
- **Secure Design**: Environment-based configuration

Performance Metrics Tracked

CPU Metrics

- User Mode Time
- System Mode Time
- Idle Time
- Interrupt Time
- CPU Utilization Percentage
- Context Switches

RAM Metrics

- Total Memory
- Available Memory
- Memory Usage Percentage
- Used Memory

- Free Memory

Technical Challenges & Solutions

1. Data Retention

Challenge: Preventing database growth **Solution**: Automatic daily data purge

```
if(_.hour==0 and _.min==0 and _.second<20):
    cur.execute(f"DELETE * FROM PHYSICALCPUSTATS WHERE IPADDRESS='{ipaddress}")
    cur.execute(f"DELETE * FROM PHYSICALCPUTIMES WHERE IPADDRESS='{ipaddress}")
    cur.execute(f"DELETE * FROM RAM WHERE IPADDRESS='{ipaddress}")
```

2. Logging & Observability

Implementation: Rotating file handler with detailed logging

```
handler = RotatingFileHandler('client.log', maxBytes=200, backupCount=0)
handler.setFormatter(
    Formatter('%(asctime)s - [line:%(lineno)d] - %(levelname)s: %(message)s')
)
```

Deployment Workflow

1. Python & Dependencies Installation
2. PostgreSQL Database Provisioning
3. Environment Configuration
4. Client Script Deployment
5. Grafana Dashboard Setup

Testing Strategy

- Unit Testing with `unittest`
- Comprehensive test coverage
- Validation of:
    - Logging mechanisms
    - IP address detection
    - Data structure integrity

Future Enhancements

- Multi-OS Support
- Advanced Anomaly Detection
- Machine Learning-based Predictive Monitoring

Technology Stack

- **Language**: Python 3.x
- **Libraries**:
  - `psutil`
  - `psycopg2`
  - `python-dotenv`
- **Database**: PostgreSQL
- **Visualization**: Grafana

Conclusion

A robust, scalable solution for comprehensive server monitoring, demonstrating expertise in Python system programming, database management, and infrastructure observability.