SysMetrics Monitoring System

Contents

1 System Overview

SysMetrics is a comprehensive system monitoring solution consisting of three main components:

1. **Metrics Collection Agent**: A lightweight agent deployed on target hosts that collects system metrics
2. **System Metrics Collector**: A Django application that stores historical metric data
3. **SysMetrics Dashboard**: A web interface for visualizing real-time and historical metrics

2 Metrics Collection Agent

2.1 Overview

The Metrics Collection Agent is a lightweight agent for collecting system metrics on Linux hosts and exposing them via a RESTful API. This agent collects detailed system metrics including:

- CPU usage and statistics
- Memory and swap usage
- Disk usage, partitions, and I/O statistics
- Process information (PID, name, CPU/memory usage, etc.)

Data is collected using Python's `psutil` library, which provides a cross-platform way to retrieve the information.

2.2 Requirements

- Python 3.8+
- Linux environment
- Network connectivity for API access

2.3 Installation

1. Clone or download the project code
2. Install dependencies:

   ```
   pip install -r requirements.txt
   ```

3. Run the agent:

   ```
   python agent/main.py
   ```

   In a browser, open URL `http://localhost:8000/metrics`

2.4 Sample Agent Response


```
{
  "timestamp": "2025-03-18 16:35:24",
  "hostname": "Host1",
  "ip_address": "127.0.1.1",
  "os_info": "Linux 5",
  "cpu": {
    "percent_usage_per_core": [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0],
    "overall_usage": 0.0909090909090909,
    "user": 0.2,
    "system": 0.3,
    "idle": 99.3,
```

```
    "cores": 22,
    "physical_cores": 11
  },
  "memory": {
    "total": 33369800704,
    "available": 30727110656,
    "used": 2189152256,
    "free": 30334898176,
    "percent_used": 7.9,
    "swap_total": 8589934592,
    "swap_used": 0,
    "swap_percent": 0
  },
  "disk": {
    "partitions": [
      {
        "device": "/dev/sdd",
        "mountpoint": "/mnt/sda1",
        "fstype": "ext4",
        "total": 1081101176832,
        "used": 18620567552,
        "free": 1007488253952,
        "percent_used": 1.8
      }
    ]
  }
}
```


3 System Metrics Collector

3.1 Overview

The System Metrics Collector is a Django application that collects and stores system metrics (CPU, memory, and disk usage) from hosts.

3.2 Features

- **Automated Collection**: Scheduled job fetches system metrics every 1 minute
- **Storage**: Stores the metrics for visualization
- **RESTful API**: Access to historical metric data
- **Time Series**: Track system performance over time

3.3 How It Works

The application consists of:

1. **Scheduler**: Background job that runs on a configurable interval
2. **API Client**: Fetches metrics from system metrics API endpoint
3. **Database Storage**: Stores key performance metrics
4. **REST API**: Provides access to the collected data

3.4 API Endpoints

- `/api/hosts/` - List all monitored hosts
- `/api/metrics/` - Access raw metrics data

3.5 Sample API Response


```
[
    {
        "id": 21,
        "hostname": "linux",
        "timestamp": "2025-04-06T18:03:14Z",
        "cpu_usage": 10.18181818181817,
        "memory_total": 33369800704,
        "memory_used": 1814437888,
        "memory_percent": 6.6,
        "disk_total": 2162202353664,
        "disk_used": 23720402944,
        "disk_percent": 1.0970482436023694
    },
    {
        "id": 20,
        "hostname": "linux",
        "timestamp": "2025-04-06T18:02:14Z",
        "cpu_usage": 13.45454545454544,
        "memory_total": 33369800704,
        "memory_used": 1823940608,
        "memory_percent": 6.7,
        "disk_total": 2162202353664,
        "disk_used": 23720402944,
        "disk_percent": 1.0970482436023694
    },
    {
        "id": 19,
        "hostname": "linux",
        "timestamp": "2025-04-06T18:01:14Z",
        "cpu_usage": 12.54545454545453,
        "memory_total": 33369800704,
        "memory_used": 1819013120,
        "memory_percent": 6.7,
        "disk_total": 2162202353664,
        "disk_used": 23720402944,
        "disk_percent": 1.0970482436023694
    }
]
```


3.6 Database Structure

The system uses two main database tables:

id	hostname	ip_address	os_info	cpu_cores
1	linux	127.0.1.1	linux	22

Figure 1: Host Table Structure

SELECT * FROM 'metrics_systemmetric' LIMIT 0,30 Execute

id	timestamp	cpu_usage	memory_total	memory_used	memory_percent	disk_total	disk_used	disk_percent	host_id
1	2025-04-06 17:26:17	96.81818181818183	33369800704	1694240768	6.3	2162202353664	23716143104	1.0968512296645765	1
2	2025-04-06 17:26:19	59.09090909090909	33369800704	1694900224	6.3	2162202353664	23716143104	1.0968512296645765	1
3	2025-04-06 17:27:17	26.81818181818182	33369800704	1687318528	6.3	2162202353664	23716143104	1.0968512296645765	1
4	2025-04-06 17:28:17	148.63636363636365	33369800704	1683443712	6.2	2162202353664	23716143104	1.0968512296645765	1
5	2025-04-06 17:28:19	36.36363636363637	33369800704	1680982016	6.2	2162202353664	23716151296	1.0968516085375337	1
6	2025-04-06 17:29:17	51.81818181818182	33369800704	1689075712	6.3	2162202353664	23716151296	1.0968516085375337	1
7	2025-04-06 17:49:28	130.45454545454544	33369800704	1801228288	6.6	2162202353664	23720378368	1.0970471069834973	1
8	2025-04-06 17:52:10	147.27272727272725	33369800704	1816006656	6.7	2162202353664	23720386560	1.0970474858564547	1
9	2025-04-06 17:52:25	122.2727272727273	33369800704	1819164672	6.7	2162202353664	23720386560	1.0970474858564547	1
10	2025-04-06 17:52:27	128.63636363636363	33369800704	1821249536	6.7	2162202353664	23720386560	1.0970474858564547	1
11	2025-04-06 17:53:14	78.63636363636364	33369800704	1813045248	6.6	2162202353664	23720386560	1.0970474858564547	1
12	2025-04-06 17:54:15	73.18181818181819	33369800704	1815617536	6.7	2162202353664	23720386560	1.0970474858564547	1

Figure 2: SystemMetric Table Structure

4 SysMetrics Dashboard

4.1 Overview

The Dashboard app is a Django-based application that collects system metrics from agents and displays them on a central dashboard. Agents return real-time data such as system stats, which is then shown on the dashboard for easy monitoring and analysis.

The code for Dashboard is located under the dashboard folder.

4.2 Prerequisites

- Python 3.8+
- Django 4.2+
- Requests library

4.3 Setup Instructions

1. Clone the repository

```
git clone <repo-url>
cd sysmetrics/dashboard
```

2. Install dependencies

```
pip install -r requirements.txt
```

3. Configure Settings
   - Ensure `http://127.0.0.1:8000/metrics` is accessible
4. Run Migrations

```
python manage.py migrate
```

5. Start the Development Server

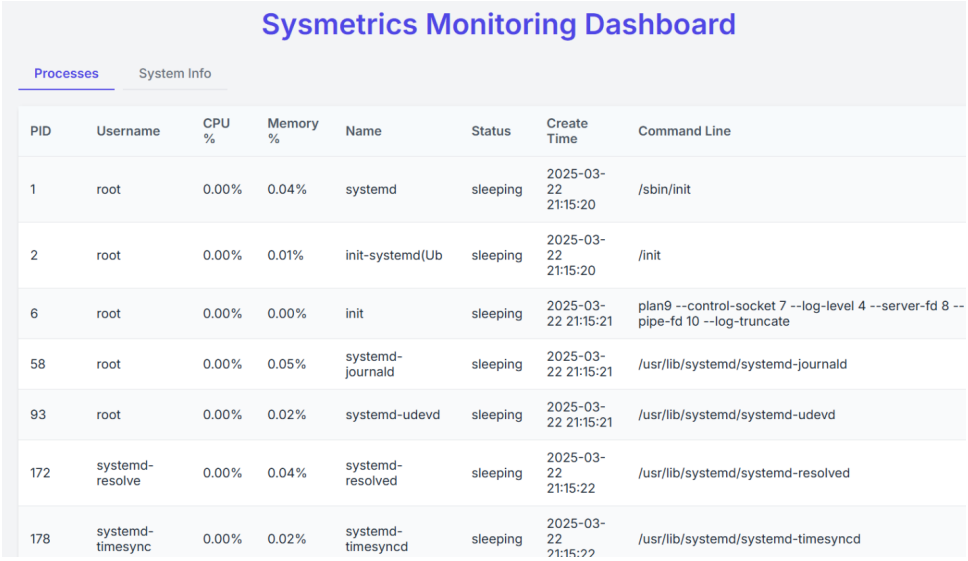
```
python manage.py runserver 0.0.0.0:7000
```

   In a browser, open URL `http://localhost:7000`

4.4 Project Structure

- `dashboard/`: Main application
  - `views.py`: Dashboard and processes views
  - `urls.py`: URL routing
  - `templates/`: HTML templates
  - `static/`: CSS and JavaScript files

4.5 Dashboard UI

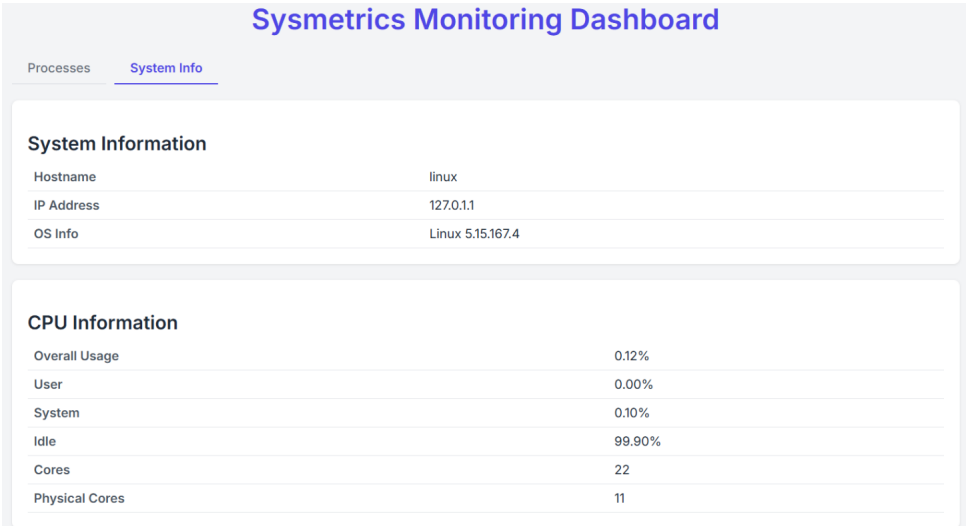


Sysmetrics Monitoring Dashboard

Processes System Info

PID	Username	CPU %	Memory %	Name	Status	Create Time	Command Line
1	root	0.00%	0.04%	systemd	sleeping	2025-03-22 21:15:20	/sbin/init
2	root	0.00%	0.01%	init-systemd(Ub	sleeping	2025-03-22 21:15:20	/init
6	root	0.00%	0.00%	init	sleeping	2025-03-22 21:15:21	plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
58	root	0.00%	0.05%	systemd-journald	sleeping	2025-03-22 21:15:21	/usr/lib/systemd/systemd-journald
93	root	0.00%	0.02%	systemd-udevd	sleeping	2025-03-22 21:15:21	/usr/lib/systemd/systemd-udevd
172	systemd-resolve	0.00%	0.04%	systemd-resolved	sleeping	2025-03-22 21:15:22	/usr/lib/systemd/systemd-resolved
178	systemd-timesync	0.00%	0.02%	systemd-timesyncd	sleeping	2025-03-22 21:15:22	/usr/lib/systemd/systemd-timesyncd

Figure 3: Processes Information UI



Sysmetrics Monitoring Dashboard

Processes System Info

System Information

Hostname	linux
IP Address	127.0.1.1
OS Info	Linux 5.15.167.4

CPU Information

Overall Usage	0.12%
User	0.00%
System	0.10%
Idle	99.90%
Cores	22
Physical Cores	11

Figure 4: System Information UI

5 Unit Test Documentation

5.1 Model Tests (test_models.py)

Tests for the database models to ensure proper data storage and relationships.

5.1.1 Host Model Tests

- Verifies Host model creation with proper attributes
- Confirms string representation method returns the hostname

5.1.2 SystemMetric Model Tests

- Verifies metric creation with proper attributes

- Confirms timestamp-based ordering
- Tests string representation method (hostname + timestamp)

```
def test_host_creation(self):
    """Test Host model creation and string representation"""
    host = Host.objects.create(
        hostname="test-server",
        ip_address="192.168.1.100",
        os_info="Ubuntu 20.04",
        cpu_cores=4
    )

    # Verify the object exists
    saved_host = Host.objects.get(id=host.id)
    self.assertEqual(saved_host.hostname, "test-server")
    self.assertEqual(saved_host.ip_address, "192.168.1.100")
    self.assertEqual(saved_host.os_info, "Ubuntu 20.04")
    self.assertEqual(saved_host.cpu_cores, 4)

    # Test string representation
    self.assertEqual(str(host), "test-server")
```

Listing 1: Sample Model Test

5.2 API Tests (test_api.py)

Tests for the REST API endpoints and serializers.

5.2.1 Host API Tests

- `test_list_hosts`: Verifies listing of all host records
- `test_retrieve_host`: Tests retrieval of a single host by ID
- `test_host_metrics_action`: Tests the custom metrics action for a host
  - Includes tests for the days parameter filtering

5.2.2 SystemMetric API Tests

- `test_list_metrics`: Verifies listing of all metric records
- `test_filter_by_hostname`: Tests filtering metrics by hostname
- `test_filter_by_days`: Tests filtering metrics by time range (days parameter)
- `test_summary_action`: Tests the summary action endpoint
  - Verifies time series and overall stats data structure

5.2.3 Serializer Tests

- `test_host_serializer`: Verifies correct Host serialization
- `test_system_metric_serializer`: Verifies correct SystemMetric serialization

```
def test_list_hosts(self):
    """Test listing all hosts"""
    url = reverse('host-list')
    response = self.client.get(url)

    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data), 2)

    # Check host data
    hosts = response.data
    self.assertEqual(hosts[0]['hostname'], self.host1.hostname)
    self.assertEqual(hosts[1]['hostname'], self.host2.hostname)
```

Listing 2: Sample API Test

5.3 Jobs Tests (test_jobs.py)

Tests for background job that collects system metrics.

- `test_run_success`: Tests successful execution of metrics collection
- `test_run_request_exception`: Tests error handling for API request failures
- `test_process_metrics_multiple_partitions`: Tests proper handling of multiple disk partitions
- `test_process_metrics_update_existing_host`: Verifies hosts are updated rather than duplicated

```
@patch('metrics.jobs.requests.get')
def test_run_success(self, mock_get):
    # Mock the API response
    mock_response = Mock()
    mock_response.json.return_value = {
        'hostname': 'test-server',
        'ip_address': '192.168.1.100',
        'os_info': 'Ubuntu 20.04',
        'timestamp': timezone.now().strftime('%Y-%m-%d %H:%M:%S'),
        'cpu': {
            'cores': 4,
            'overall_usage': 0.25  # 25%
        },
        'memory': {
            'total': 8589934592,  # 8GB
            'used': 4294967296,   # 4GB
            'percent_used': 50.0
        },
        'disk': {
            'partitions': [
                {
                    'mount_point': '/',
                    'total': 107374182400,  # 100GB
                    'used': 32212254720,    # 30GB
                    'percent': 30.0
                }
            ]
        }
    }
    mock_get.return_value = mock_response

    # Execute the job
    result = self.job.run()

    # Verify job executed successfully
    self.assertTrue(result)
```

Listing 3: Sample Job Test

5.4 Views Tests

Tests for view functions to ensure proper rendering and context data.

- `test_index_view_successful_api`: Tests index view with successful API response
- `test_index_view_api_failure`: Tests index view with API failure handling
- `test_processes_view_successful_api`: Tests processes view with successful API response
- `test_processes_view_api_failure`: Tests processes view with API failure handling

```
@patch('core.views.requests.get')
def test_index_view_successful_api(self, mock_get, mock_metrics_response, client):
    # Configure the mock to return a successful response
    mock_response = Mock()
    mock_response.json.return_value = mock_metrics_response
    mock_get.return_value = mock_response
```

```
    # Get the index page
    url = reverse('dashboard_index')
    response = client.get(url)

    # Verify the response
    assert response.status_code == 200
    assert 'metrics' in response.context
    assert response.context['metrics'] == mock_metrics_response
```

Listing 4: Sample View Test

5.5 Running Tests

To run the tests:

```
# Run view tests
pytest core/tests/test_views.py -v

# Run metrics tests
pytest metrics/tests/ -v
```

6 API Testing

6.1 API Endpoints for Testing

6.1.1 Agent Metrics Endpoints

- **GET /metrics** - Returns all system metrics for a host

6.1.2 Host Endpoints

- **GET historical/api/hosts/** - List all hosts
- **GET historical/api/hosts/metrics/** - Get historical metrics for a specific host

6.1.3 REST Client Test Results

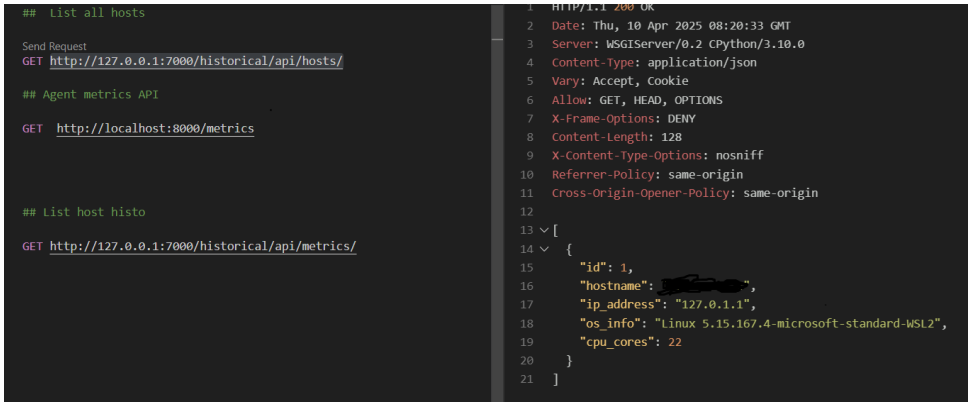


Figure 5: Get all hosts

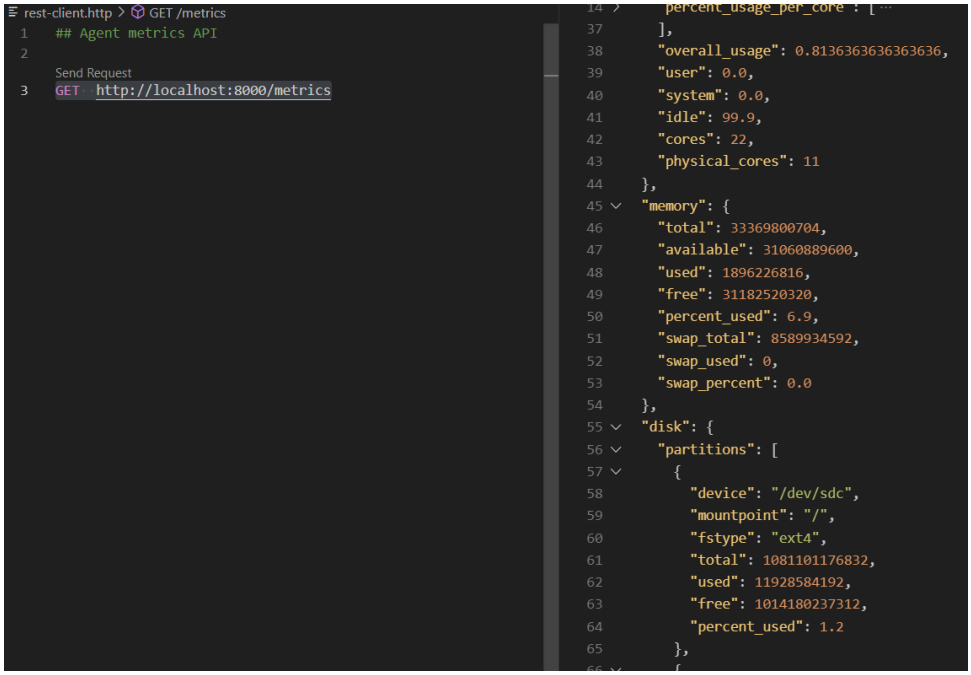


Figure 6: Agent Metrics API

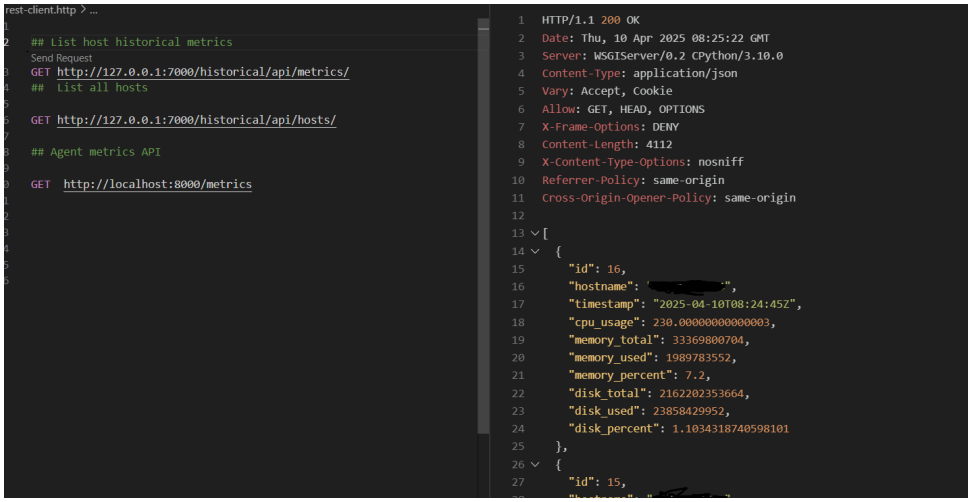


Figure 7: Host Historical Metrics

7 End-to-End Testing Scenarios

7.1 API Functionality Testing

1. List all hosts
2. Retrieve host metrics from agent API
3. Get historical metrics for that host

8 Dashboard UI End-to-End Testing

8.1 Test Scenarios

8.1.1 Dashboard Home Page

1. Verify dashboard loads correctly
2. Confirm metrics cards display current CPU usage, memory usage, and disk usage
3. Verify time-series charts load and display data
4. Test refresh functionality updates metrics data

8.1.2 System Information Tab

1. Verify system information tab loads correctly
2. Confirm CPU information section displays correctly
3. Confirm memory information section displays correctly
4. Confirm disk information displays for all partitions

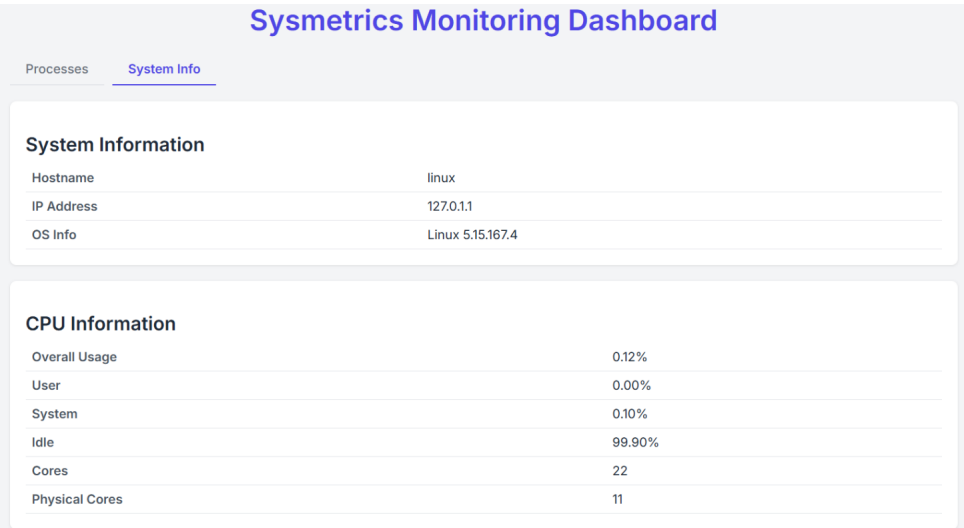
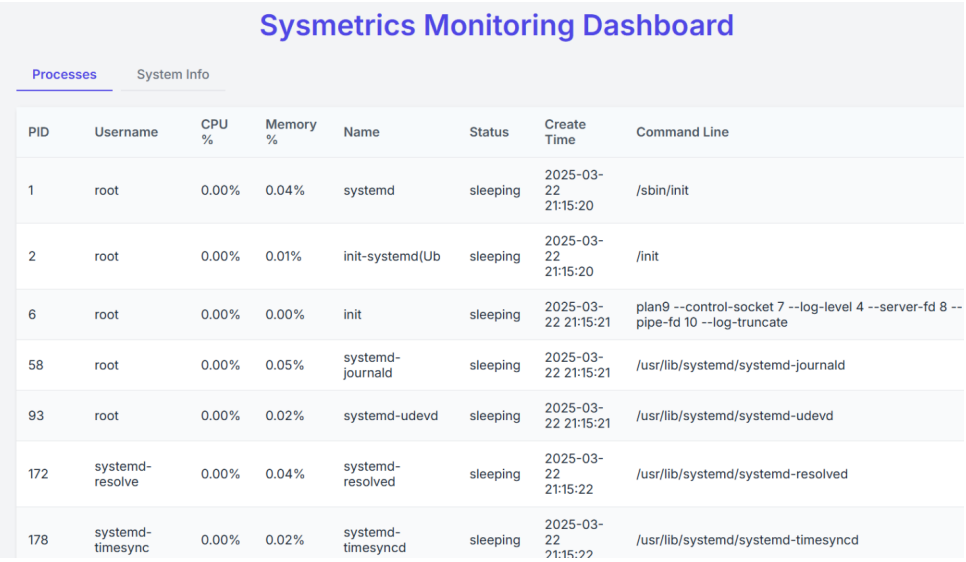


Figure 8: System Information Tab Interface

8.1.3 Processes Tab

1. Verify processes tab loads correctly
2. Confirm all processes are listed in a table



Sysmetrics Monitoring Dashboard

Processes System Info

PID	Username	CPU %	Memory %	Name	Status	Create Time	Command Line
1	root	0.00%	0.04%	systemd	sleeping	2025-03-22 21:15:20	/sbin/init
2	root	0.00%	0.01%	init-systemd(Ub	sleeping	2025-03-22 21:15:20	/init
6	root	0.00%	0.00%	init	sleeping	2025-03-22 21:15:21	plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
58	root	0.00%	0.05%	systemd-journald	sleeping	2025-03-22 21:15:21	/usr/lib/systemd/systemd-journald
93	root	0.00%	0.02%	systemd-udevd	sleeping	2025-03-22 21:15:21	/usr/lib/systemd/systemd-udevd
172	systemd-resolve	0.00%	0.04%	systemd-resolved	sleeping	2025-03-22 21:15:22	/usr/lib/systemd/systemd-resolved
178	systemd-timesync	0.00%	0.02%	systemd-timesyncd	sleeping	2025-03-22 21:15:22	/usr/lib/systemd/systemd-timesyncd

Figure 9: Processes Tab Interface

9 Notes and Recommendations

- Ensure your metrics API returns data in the expected JSON format
- When troubleshooting, check the agent logs first, then the collector, and finally the dashboard