Project: An SRE-Driven Framework for Detecting and Prioritizing Security Incidents Using Reliability Metrics in Cloud Native Systems
Reporting Period: January 27, 2025 - February 10, 2025 (2 Weeks)
Team Member: Manula Hindurangala – 300390724

1. Work Date/Hours Logs

Date	Number of Hours	Description of Work Done
28-Jan-25	3	**Phase 1**: Environment Setup. Researched and finalized Minikube architecture. Prepared Linux VM (Ubuntu 22.04) with required resources (8GB RAM, 4 vCPUs). Installed and configured Docker, verified with hello-world.
30-Jan-25	3.5	**Phase 1 Continued**: Toolchain Installation. Installed Minikube, kubectl, and Helm. Added essential Helm repositories. Started the Minikube cluster using --driver=docker with metrics-server, dashboard, and ingress addons. Verified cluster health with kubectl get pods --all-namespaces.
03-Feb-25	3.5	**Phase 2 & 3**: Project & Backend Foundation. Created the complete ~/sre-security-lab project directory structure. Set up Python virtual environment and created requirements.txt. Developed the core Flask application (app.py) with health, metrics, and scenario API endpoints. Created the scenarios.json database with three defined scenarios (DDoS, Brute Force, Config Error).
06-Feb-25	4.5	**Phase 4**: Frontend Initialization & Troubleshooting. Initialized the React application with create-react-app. Installed UI and charting dependencies (axios, Material-UI, Chart.js). Resolved initial environment and dependency conflicts.
09-Feb-25	4.5	Documentation & Repository Setup. Organized all completed work. Created comprehensive documentation for the setup process.

2. Summary Description of Work Done During This Reporting Period

Over the past two weeks, I have successfully built the complete foundation for the SRE Security Research Lab. The work focused on establishing a production-like Kubernetes environment on a Linux VM and developing the core applications for security research. I began by making a key architectural decision to use Minikube's --driver=docker for optimal isolation and simplicity. After setting up the VM, I installed the entire toolchain (Docker, Minikube, kubectl, Helm) and launched a fully functional Kubernetes cluster with critical addons. I then developed the backend Flask API, which includes a Prometheus metrics endpoint and a simulation engine for three distinct security/operational scenarios.

Concurrently, I initialized the React frontend dashboard. The final phase involved consolidating all code, configurations, and setup documentation into the project's GitHub repository. No major blockers were encountered; the work progressed systematically according to the planned phases.

3. Repo Check-in of Implementation Completed

The following implementation has been checked into the Implementation/sre-security-lab/ directory of the shared GitHub repository:

- Infrastructure & Configuration:
  docs/setup_guide.docx (Complete setup instructions for Phases 1-4)
- docs/driver_decision.docx (Architecture analysis)
- Backend Application (Complete):

  backend/src/app.py (Full Flask application with API)

  backend/requirements.txt (Python dependencies)

  backend/scenarios.json (Three defined scenarios)

- Frontend Application (Initialized):

  frontend/package.json (React & dependency configuration)

4. AI Use Section

AI Tool Name	**Version, Type**	**Account**	**Specific feature for which the AI tool was used**	**Value Addition**
DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Architecture Decision Analysis - Comparative evaluation of Minikube's --driver=none vs --driver=docker for VM-based Kubernetes setup.	Critical Decision Making - Analyzed AI's pros/cons list and selected --driver=docker based on research priorities (isolation, reproducibility) rather than raw performance metrics. Designed the final VM resource allocation strategy.
DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Code Template Generation - Generated initial Flask application template (app.py) with basic structure, Prometheus metrics endpoints, and REST API framework.	Custom Implementation & Research Context - Significantly modified the template to include three specific security/operational scenarios. Added the scenario simulation logic, custom metric calculations, and research-focused data structures tailored to SRE security analysis.

DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Infrastructure Setup Scripts - Provided complete command sequences for installing Docker, Minikube, kubectl, Helm, and system dependencies on Ubuntu 22.04.	System Integration & Validation - Tested all commands in the actual VM environment, resolved environment-specific issues (permissions, networking), and created a verified, sequential setup guide. Integrated all components into a cohesive system design.
DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Troubleshooting & Debugging - Diagnosed and provided solutions for Ingress controller pod failures (ImagePullBackOff) and Node.js version conflicts during React setup.	Problem Resolution & Adaptation - Applied the suggested fixes, but also identified root causes and implemented preventative measures. Created comprehensive troubleshooting documentation based on actual issues encountered during implementation.
DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Documentation Structuring - Assisted in organizing the progress report according to the professor's template and helped format technical documentation in markdown.	Content Creation & Quality Enhancement - Transformed AI's structural suggestions into detailed, professionally formatted documents. Added practical implementation notes, verification steps, and research context missing from AI-generated outlines. Created documentation optimized for team collaboration.
DeepSeek AI Chat Assistant	Web interface, Latest version (Feb 2025)	Free tier	Configuration File Creation - Generated package.json with React dependencies and scenarios.json database structure for security scenarios.	Customization & Research Design - Modified dependency versions to ensure compatibility. Transformed the generic JSON structure into three detailed, research-relevant security scenarios with specific metrics tailored for SRE analysis. Added comprehensive descriptions and classification logic.

5. Individual Timeline for Next Steps Planned

1. Week of Feb 10: Complete the frontend React components (UI, charts, API service layer) and containerize both applications by writing and testing Dockerfiles.

2. Week of Feb 17: Develop the Kubernetes Deployment and Service YAML manifests. Subsequently, create and test the Helm charts for automated deployment. Deploy the full stack to the Minikube cluster.

3. Week of Feb 24: Implement the security simulation engine and begin initial data collection and analysis for the research project.

Appendix

1. Write a Flask app with a /metrics endpoint that exposes Prometheus metrics.

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from prometheus_client import generate_latest, Counter, Histogram, REGISTRY
import json
import time
import numpy as np

app = Flask(__name__)
CORS(app)

# Prometheus metrics
REQUEST_COUNT = Counter('http_requests_total', 'Total HTTP Requests', ['method', 'end
point', 'status'])
REQUEST_LATENCY = Histogram('http_request_duration_seconds', 'HTTP request latency',
['endpoint'])

# Load scenarios
with open('scenarios/scenarios.json', 'r') as f:
    scenarios = json.load(f)

@app.route('/api/health', methods=['GET'])
def health():
    return jsonify({"status": "healthy", "timestamp": time.time()})

@app.route('/api/metrics', methods=['GET'])
def metrics():
    return generate_latest(REGISTRY), 200, {'Content-Type': 'text/plain'}

@app.route('/api/scenarios', methods=['GET'])
def get_scenarios():
    REQUEST_COUNT.labels(method='GET', endpoint='/scenarios', status='200').inc()
    with REQUEST_LATENCY.labels(endpoint='/scenarios').time():
        return jsonify(scenarios)
```

```python
@app.route('/api/simulate/<scenario_id>', methods=['POST'])
def simulate_scenario(scenario_id):
    REQUEST_COUNT.labels(method='POST', endpoint='/simulate', status='200').inc()
    with REQUEST_LATENCY.labels(endpoint='/simulate').time():
        scenario = next((s for s in scenarios if s['id'] == scenario_id), None)
        if not scenario:
            return jsonify({"error": "Scenario not found"}), 404

        # Simulate some processing
        time.sleep(0.1)

        # Add simulation results
        result = {
            "scenario": scenario['name'],
            "type": scenario['type'],
            "metrics": scenario['metrics'],
            "simulation_id": f"sim_{int(time.time())}",
            "timestamp": time.time(),
            "analysis": {
                "risk_level": "high" if scenario['type'] == 'security' else "medium",
                "recommendations": ["Increase monitoring", "Review logs", "Check reso
urce usage"]
            }
        }
        return jsonify(result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=False)
```

The exact `/metrics` endpoint code from your Flask app is:

python

```python
@app.route('/api/metrics', methods=['GET'])
def metrics():
    return generate_latest(REGISTRY), 200, {'Content-Type': 'text/plain'}
```

2. What is the difference between Minikube's docker driver and none driver for a setup inside a Linux VM?

Minikube's docker driver vs none driver for a Linux VM setup:

`--driver=none` **(Direct/Native Installation):**

- Runs Kubernetes components (kubelet, apiserver, etcd, scheduler) **directly on the VM's host operating system** as native processes
- Requires system dependencies (conntrack, socat, ebtables, etc.) to be installed on the VM
- Provides **production-like architecture** with direct hardware access
- More complex to set up and troubleshoot
- Better for accurate performance metrics and system-level research
- Example: `sudo minikube start --driver=none --cpus=4 --memory=6144`

`--driver=docker` **(Containerized Installation):**

- Runs Kubernetes components **inside Docker containers** on the VM
- Uses Docker as both container runtime and virtualization layer
- **Cleaner isolation** - entire Kubernetes cluster runs in containers
- **Easier setup** with fewer system dependencies
- **Nested architecture** (Docker-in-VM, containers-in-containers)
- Less accurate resource metrics due to abstraction layers
- Example: `minikube start --driver=docker --cpus=4 --memory=6144`

Key Difference: `none` driver runs Kubernetes natively on VM OS (production-like), while `docker` driver runs Kubernetes in Docker containers on the VM (easier isolation).

3. Create a JSON structure for storing Kubernetes security scenario data.

```
[
  {
    "id": "dos-attack",
    "name": "Layer 7 DDoS Attack",
    "type": "security",
    "metrics": {
      "latency_p99": 1500,
      "error_rate": 0.35,
      "request_rate": 5200,
```

```
      "cpu_utilization": 0.92,
      "memory_utilization": 0.85,
      "network_in": 450,
      "network_out": 380
    },
    "description": "Simulated HTTP flood attack causing resource exhaustion"
  },
  {
    "id": "brute-force",
    "name": "Credential Brute Force Attack",
    "type": "security",
    "metrics": {
      "latency_p99": 420,
      "error_rate": 0.45,
      "request_rate": 1800,
      "auth_failures": 1250,
      "unique_ips": 220,
      "success_rate": 0.02
    },
    "description": "Multiple failed login attempts from diverse IPs"
  },
  {
    "id": "config-error",
    "name": "Configuration Error",
    "type": "operational",
    "metrics": {
      "latency_p99": 820,
      "error_rate": 0.22,
      "request_rate": 950,
```

```
            "cpu_utilization": 0.48,
            "memory_utilization": 0.52
        },
        "description": "Recent configuration change causing degradation"
    }
]
```

4. List the commands to install Helm and add the common repositories.

```
# Download Helm installer
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

# Install Helm
chmod 700 get_helm.sh
./get_helm.sh

# Verify installation
helm version

# Add Helm repositories (optional but recommended)
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
```