✅ DISSANAYAKE E.G.M — Deployment & Evaluation Engineer

This version will be:

✓ Fully aligned with the *actual state* of SmartOps (your orchestrator, telemetry stack, ERP simulator, chaos features, RBAC status).
✓ Structured professionally like an industry SRE / DevOps implementation plan.
✓ No illusions — only tasks that match what exists or is achievable within your architecture.
✓ Clean, clear, and ready to paste into your project report.

🚀 DISSANAYAKE E.G.M — COMPREHENSIVE IMPLEMENTATION PLAN

Role: Deployment, Observability & Resilience Validation Lead

Dissanayake owns the **infrastructure backbone** of SmartOps — the Kubernetes platform, CI/CD automation, telemetry stack, chaos testing suite, and evaluation pipelines.
His responsibility is to ensure SmartOps can be **deployed, observed, stress-tested, and benchmarked reliably** in a real cluster.

Workload share: **27–30%**

SECTION 1 — ROLE OVERVIEW & OBJECTIVE

Dissanayake is responsible for:

✓ Building and maintaining the Kubernetes platform

✓ Implementing CI/CD workflows

✓ Deploying all SmartOps services (orchestrator, ERP simulator, AI agents, policy engine)

✓ Integrating telemetry: Prometheus, Loki, Grafana, OTel Collector, Tempo

✓ Installing and running chaos experiments

✓ Producing MTTR, SLO, SLA, recovery KPIs

✓ Final benchmarking for viva presentation

He ensures the system operates *continuously*, auto-heals under stress, and provides complete visibility from metrics to logs to traces.

SECTION 2 — ENVIRONMENT & PLATFORM SETUP (WEEKS 1–2)

Tasks

- Create GitHub organization (smartops-ai) and configure:
  - CODEOWNERS
  - main branch protection
  - PR templates
  - issue templates
  - Actions secrets (KUBECONFIG_DEV, GHCR_PAT, etc.)
- Provision Kubernetes cluster (Docker Desktop / Minikube / K3s / cloud)
- Create namespaces:

```
smartops-dev
smartops-stage
smartops-prod
```

- Initialize the Helm umbrella chart in:

```
platform/helm/smartops/
```

containing:

- Prometheus stack
- Loki
- Tempo
- OTel Collector
- ERP Simulator chart
- Orchestrator chart (already exists)

Collaboration

- Peiris → align orchestrator RBAC & service account permissions
- Team → ensure secrets and endpoints shared

Acceptance

✓ Helm install succeeds
✓ Grafana dashboard accessible
✓ CI skeleton workflow runs

SECTION 3 — CI/CD PIPELINE IMPLEMENTATION (WEEKS 1–3)

CI/CD Responsibilities

- Build Docker images → Push to GHCR
- Deploy orchestrator, ERP simulator, AI agents, policy-engine
- Automatic promotion workflow (dev → stage → prod)
- Rollback capability

Tasks

- Implement reusable GitHub Actions:

```
ci.yml
deploy-dev.yml
promote-stage.yml
promote-prod.yml
```

- Add artifact caching, parallel jobs
- Validate Helm lint + Kubernetes manifest tests
- Configure semantic versioning (v0.1.0, v0.1.1, etc.)
- Add rollback logic:

```
helm upgrade --install smartops . --atomic --wait
```

Collaboration

- Kulathunga → integrate model unit tests, model registry build
- Gunarathne → policy engine test suite
- Peiris → orchestrator build + startup validation

Acceptance

✓ Merge to `main` → auto deploys DEV
✓ Tag release → deploys to STAGE
✓ Rollback works via GitHub Actions

SECTION 4 — TELEMETRY & OBSERVABILITY STACK (WEEKS 2–4)

Deploy Observability Components

- Prometheus Operator
- Loki Stack (log aggregation)
- OTel Collector (trace pipeline)
- Tempo (distributed tracing backend)
- Grafana (visualization layer)

Configure Data Sources

- Prometheus scrape targets:
  - orchestrator
  - ERP simulator
  - future AI services
  - kube-state-metrics
- Loki logs:
  - orchestrator logs
  - pod crash logs
- Tempo traces via OTLP
- Dashboards using JSON or GUI

Collaboration

- Peiris → validate orchestrator metrics & tracing
- Kulathunga → export AI agent metrics
- Gunarathne → add policy engine metrics

Acceptance

✓ "MTTR", "Error rate", "Closed-Loop Actions" dashboards working
✓ Logs + metrics + traces linkable by trace ID
✓ Alerts firing: pod restart, SLA breach, anomaly spike

SECTION 5 — CHAOS ENGINEERING SETUP (WEEKS 3–4)

Chaos Tools

- Chaos Mesh
  (Or Chaos Toolkit depending on resource availability)

Experiments Owned

- Podkill: kill orchestrator or ERP simulator pod
- CPU stress: 80–100% CPU
- Memory leak: 500–800MB leak
- Network delay: 200–1000ms
- Packet loss: 10–30%
- Combined scenarios: multiple chaos conditions

How chaos supports SmartOps

Chaos validates:

- AI anomaly detection
- RCA mapping
- Closed-loop actions
- Kubernetes recovery
- MTTR metrics

Collaboration

- Peiris → orchestrator failure scopes
- Kulathunga → training data windows for chaos
- Gunarathne → policy rule interaction under chaos

Acceptance

✓ All chaos experiments execute safely
✓ Recovery is measurable (MTTR shown on Grafana)
✓ Logs stored for evaluation

SECTION 6 — DEPLOYMENT & RELEASE MANAGEMENT (WEEKS 4–10)

Responsibilities

- Maintain Helm values per environment
- Ensure safe promotions:
  - dev → stage → prod
- Validate Kubernetes health gates
- Maintain image version chart in GitHub Actions

Promotion Policy

- Stage deploy only allowed if:
    - No failing workloads
    - Prometheus alerts in green
    - MTTR < threshold
- Prod deploy only allowed if:
    - Stage “golden window” of 72 hours passes
    - No anomaly spikes
    - No persistent alerts

Collaboration

- Gunarathne → deploy policy-engine subchart
- Peiris → ensure orchestrator scales/restarts correctly

Acceptance

✓ Prod deployment successful with no failing pods
✓ Rollout verified with Grafana + Prometheus

SECTION 7 — VISUALIZATION & OPS CONSOLE (WEEKS 8–14)

Dashboards Required

1. **Cluster Overview**
    - CPU, RAM, nodes, pod restarts
2. **Orchestrator Actions Panel**
    - scale, restart, patch
    - closed-loop duration
    - queue depth
3. **AI Agent Metrics**
    - anomaly scores
    - RCA confidence
    - model drift
4. **Policy Engine Metrics**
    - guardrail hits
    - policy evaluation latency
    - approved vs blocked actions
5. **Chaos vs Auto-Heal Timeline**
    - chaos event
    - anomaly detection
    - RCA
    - orchestrator action
    - verification

Alerts

- MTTR > threshold
- CrashLoopBackOff
- Missing metrics from orchestrator
- AI inference latency too high
- Policy conflicts too frequent

Acceptance

✓ Dashboards auto-refresh
✓ Alerts trigger within <60s
✓ All components are observable

SECTION 8 — RESILIENCE EVALUATION & BENCHMARKING (WEEKS 11–14)

Design Evaluation Test Matrix

Dimensions:

- Chaos Type
- AI Model Variant
- Policy Engine Rule Set
- Workload Intensity

Metrics to Capture

- MTTR reduction percentage
- Auto-heal success rate
- Correct vs incorrect RCA
- SLO adherence
- SLA uptime
- Closed-loop duration
- Action success vs failure
- Number of guardrail blocks

Output

Produce:

```
/tests/chaos/results/
/docs/reports/evaluation.pdf
```

Acceptance

✓ MTTR improved ≥ 40%
✓ Auto-heal ≥ 90% accuracy
✓ SLA ≥ 99% maintained

SECTION 9 — HARDENING & SECURITY (WEEKS 14–16)

Tasks

- Tighten RBAC for orchestrator, AI agents, policy engine
- Apply network policies limiting:
  - namespace egress
  - cross-service calls
- Add resource limits & HPA targets
- Enable image signing (cosign)
- Add vulnerability scanning
- Add promotion SLO gates (cannot deploy if SLO < 98%)

Acceptance

✓ No unauthorized API paths above allowed verbs
✓ 72-hour staging observation with no critical alerts
✓ Prod promotions gated by SLO score

SECTION 10 — DOCUMENTATION & KNOWLEDGE TRANSFER

Must Produce

- `/docs/runbooks/*`
- `/docs/deployment_guide.md`
- `/docs/chaos_manual.md`
- “How to Reproduce Evaluation” guide
- Grafana dashboard index

Acceptance

✓ All docs reproducible
✓ Fresh-cluster deploy validated
✓ Viva-ready

SECTION 11 — KPIs (FOR DISSANAYAKE’S EVALUATION)

KPI	Target	Tool
Deployment success rate	≥ 95%	GitHub Actions
Chaos test reliability	≥ 90%	Chaos Mesh
MTTR reduction	≥ 40%	Grafana MTTR panel
Dashboard accuracy	≥ 90%	Grafana vs Loki logs
SLA adherence	≥ 99%	Prometheus Query

SECTION 12 — INTERFACES & TEAM COLLABORATION

Collaborator	Interaction
Peiris P.V.G	consumes logs/metrics; validates orchestrator actions under chaos
Kulathunga K.A.K.M	supplies AI agent metrics and model performance
Gunarathne M.D.C.H	integrates policy metrics, guardrails, action plans
Supervisor	final evaluation & deployment quality assurance

SECTION 13 — FINAL DELIVERABLES (OWNED ENTIRELY BY DISSANAYAKE)

1. **Helm Umbrella Chart & Values**
2. **CI/CD Workflow Stack**
3. **Chaos Testing Suite**
4. **Grafana Dashboards**
5. **Evaluation Reports**
6. **Runbooks & Deployment Guides**
7. **Stage/Prod Hardening Checklist**

⭐ FINAL SUMMARY

Dissanayake ensures SmartOps is:

- Deployable
- Observable
- Chaos-validated
- Benchmarkable
- Secure
- Production-ready

His contributions form the **operational backbone** allowing Peiris (orchestrator), Kulathunga (AI), and Gunarathne (policy engine) to deliver a fully autonomous, observable, and resilient platform.