Infrastructure Cost & Efficiency Analyzer (ICEA)

MVP Specification + Developer Design Specs

Prepared for: Ian Salandy — February 22, 2026

Goal: Launch a paid diagnostic tool that produces a dollar-denominated optimization report for Spark (and later Trino) workloads.
Fastest revenue path: Sell a $500–$3,000 “single analysis report” first; then productize into self-serve SaaS.
Primary outputs: (1) Efficiency score, (2) monthly waste estimate, (3) recommended configuration, (4) projected savings, (5) risk notes.

Initial workload focus	Spark on Kubernetes / EMR / Databricks
Target buyers	Data Engineering, Platform Engineering, FinOps, CTO/VP Eng
MVP inputs	Cluster + executor config + runtime + jobs/day + pricing
MVP delivery	Web form + PDF report generator (tool-assisted, human-reviewed)
Monetization	Paid report first, then subscription + enterprise license

Note: KPI99 currently has no customers; this spec is designed to generate revenue from net-new leads without requiring existing references.

Contents

1. Product definition and positioning

ICEA is a lightweight diagnostic tool that converts infrastructure configuration into business-facing outputs: waste, risk, and recommended changes. The key differentiator is financial impact (cost-to-serve), not just technical tuning. The MVP is intentionally narrow: Spark executor packing and cluster utilization, with support for Kubernetes, EMR, and Databricks pricing inputs.

Value proposition

- Identify 10–40% infrastructure waste caused by executor packing and configuration mismatch.
- Produce a PDF report that can be shared with engineering leadership and finance.
- Recommend a concrete executor/node configuration and quantify savings.
- Optionally flag high-risk patterns (OOM likelihood, GC pressure, shuffle spill risk) as advisory notes.

Non-goals for MVP

- No deep Spark event-log ingestion in v1 (add later).
- No automated code or query rewriting in v1.
- No always-on monitoring agent in v1.

2. MVP scope and user journeys

Primary journey (paid report)

- **Lead** enters configuration and workload basics (5 minutes).
- **Tool** computes efficiency, waste estimate, and recommendations.
- **Operator review** (you) sanity-checks inputs and adds 2–5 bullet insights.
- **Tool** generates a branded PDF report and delivers it to the lead.
- **Upsell**: offer implementation support or subscription monitoring.

Secondary journey (self-serve preview)

- Anonymous user runs a limited calculation (no PDF) to see potential waste range.
- Prompt user to unlock a full report (payment) and provide contact details.

3. Data inputs and assumptions

Required inputs (MVP form)

Field	Notes / examples
Cloud / environment	AWS | Azure | GCP | On-prem (pricing manual)
Node instance type	e.g., r6g.4xlarge / m5.2xlarge (optional if pricing manual)
Node vCPU and memory	cores_per_node, memory_gb_per_node
Node hourly cost	USD per hour per node (manual entry)
Cluster size	node_count
Executor cores and memory	executor_cores, executor_memory_gb
Job runtime	avg_runtime_minutes per run
Job frequency	jobs_per_day (or runs/week)
Utilization posture	exclusive cluster vs shared (optional v1)

Assumptions (v1): The model uses conservative packing heuristics (floor-based) and treats the dominant constraint (CPU or memory) as the waste driver. For shared clusters, add an optional utilization factor to avoid overstating waste.

4. Core calculations (efficiency + waste + savings)

Notation

Let **C** be cores_per_node, **M** be memory_gb_per_node, **c** be executor_cores, **m** be executor_memory_gb. Let **P** be node_hourly_cost, **N** be node_count, **R** be avg_runtime_minutes, and **J** be jobs_per_day.

MVP formula block (reference implementation)

```
# Executors per node (dominant by CPU/memory)
e_cpu = floor(C / c)
e_mem = floor(M / m)
executors_per_node = min(e_cpu, e_mem)

cpu_utilization = (executors_per_node * c) / C
mem_utilization = (executors_per_node * m) / M

cpu_waste = 1 - cpu_utilization
mem_waste = 1 - mem_utilization

# Cost model
hourly_cluster_cost = P * N
daily_cost = hourly_cluster_cost * (R / 60.0) * J
waste_cost_daily = daily_cost * max(cpu_waste, mem_waste)
waste_cost_monthly = waste_cost_daily * 30
```

Efficiency score (0–100)

Compute a simple score that is stable and explainable: **score = round(100 * (1 - max(cpu_waste, mem_waste)))**. Optionally apply penalties for risk flags (see Section 5).

5. Recommendation engine (rules)

The MVP recommendation engine is rule-based. It explores a small grid of candidate executor sizes and selects the configuration that minimizes waste while respecting guardrails (minimum memory, maximum executors, etc.).

Candidate search space

- Executor cores candidates: [1, 2, 3, 4, 5, 6, 8] (filter by C).
- Executor memory candidates: step in 1 GB or 2 GB increments within [4 GB, M].
- Optionally reserve node overhead: subtract 1 core and 2–4 GB for daemon/system overhead.

Guardrails (v1)

- Minimum executor memory: 6–8 GB default (configurable).
- Maximum executors per node: cap at 8–12 to avoid overhead (configurable).
- If e_mem == 0 or e_cpu == 0, mark configuration invalid.
- If memory waste is low but CPU waste high, consider increasing cores per executor; if reverse, adjust memory.

Optional risk flags (advisory notes)

- High OOM likelihood: executor_memory_gb too low for workload class (manual toggle in MVP).
- GC pressure: heap fraction too high vs executor memory (later integrate heuristics).
- Shuffle spill risk: small memory + high shuffle workloads (later integrate Spark metrics).

6. Report output format (PDF)

Report sections (1–3 pages typical)

- Executive summary (waste estimate, score, recommended change).
- Current vs recommended configuration (side-by-side table).
- Cost model assumptions (inputs, runtime, frequency, pricing).
- Savings projection (daily + monthly) and sensitivity note.
- Engineering notes (2–5 bullets; explain constraints and trade-offs).

PDF layout specs

Spec	Value
Page size	US Letter (8.5 x 11 in)
Margins	0.75 in left/right, 0.8 in top, 0.75 in bottom
Typography	Headings: Helvetica-Bold; Body: Helvetica; Code: Courier

Base font sizes	Title 28; H1 16; H2 12.5; Body 10.5; Small 9
Color palette	Dark #111827; Muted #6B7280; Accent #1F4DFF; Light bg #F6F7F9
Tables	Header row dark background; grid lines #D1D5DB; padding 6–8 px
Callouts	Light blue bg #F3F8FF with border #C7DBFF; padding 10 px
Footer	Page number + doc name + date

7. System architecture

Keep the MVP architecture minimal. The product is a calculator and a report generator, not a monitoring platform. Start with a single backend service and a simple frontend form.

Components

- **Frontend**: single-page form with validation.
- **Backend**: Python service (FastAPI recommended) exposing compute + report endpoints.
- **Core engine**: pure functions (no I/O) to compute packing, waste, and recommendations.
- **Report generator**: deterministic PDF generation (ReportLab) for consistent layout.
- **Storage** (optional): store submissions + generated PDFs (S3/Blob storage).

Module layout (Python)

```
icea/
__init__.py
models.py # input/output types
packing.py # executor packing + utilization
cost_model.py # cost calculations and projections
recommend.py # search + guardrails
report/
templates.py # section builders
pdf.py # PDF rendering
api.py # FastAPI routes (optional)
tests/
test_packing.py
test_cost_model.py
```

8. API design (optional for MVP)

Endpoint	Description
POST /v1/analyze	Returns computed metrics + recommendation JSON
POST /v1/report	Returns PDF (application/pdf) for given inputs
GET /v1/health	Health check

Sample request JSON

```
{
"cloud": "aws",
"node": {"cores": 16, "memory_gb": 64, "hourly_cost_usd": 0.92, "count": 10},
"executor": {"cores": 4, "memory_gb": 16},
"workload": {"avg_runtime_minutes": 18, "jobs_per_day": 40},
"assumptions": {"reserve_cores": 1, "reserve_memory_gb": 4}
}
```

9. Frontend design specs

The frontend should feel like a premium calculator: minimal inputs, instant results preview, and a strong call-to-action to unlock a full PDF report.

Form sections

- **Environment**: cloud, node hourly cost (manual), node cores/memory, node count.
- **Executors**: executor cores, executor memory, overhead reserves toggle.
- **Workload**: avg runtime minutes, jobs per day.
- **Contact** (paid report): name, email, company, optional notes.

UX requirements

- Inline validation and friendly error messages.
- Live preview: efficiency score + estimated monthly waste range.
- One primary CTA: **Generate Full PDF Report** (gated behind payment or lead capture).
- Mobile responsive (form fields stack vertically).

10. Observability, security, and privacy

- Do not require sensitive credentials in MVP. Prefer manual pricing inputs.
- If storing submissions, encrypt at rest and set retention (e.g., 30–90 days).
- Log only request IDs and coarse metrics; avoid logging full inputs by default.
- Add rate limiting to public endpoints.
- Include a disclaimer: estimates are directional and depend on workload behavior.

11. Pricing and packaging

Package	Price guidance
Single Analysis Report (fastest revenue)	$500 to $3,000 per report (tool + expert review)
Self-serve subscription (later)	$49 to $199/month (limited reports, saved scenarios)
Enterprise license (later)	$5k to $25k/year (SSO, teams, custom guardrails)

12. Build plan (14-day launch)

When	Deliverable
Days 1–2	Implement core calculations + unit tests.
Days 3–4	Implement recommendation grid search + guardrails.

Days 5–6	Implement PDF generator and report template.
Days 7–8	Build frontend form + preview results.
Days 9–10	Deploy (single service) + basic analytics.
Days 11–14	Start outbound: LinkedIn messages + 3 demo reports for proof.

Appendix A: Reference formulas

Use this appendix as the single source of truth for engineering implementation. Values are floats unless stated.

Metric	Formula
Executors per node	executors_per_node = min(floor((C - reserve_cores) / c), floor((M - reserve_mem) / m))
CPU utilization	cpu_util = (executors_per_node * c) / (C - reserve_cores)
Memory utilization	mem_util = (executors_per_node * m) / (M - reserve_mem)
Waste driver	waste = max(1 - cpu_util, 1 - mem_util)
Hourly cluster cost	cluster_cost_hr = P * N
Daily cost	daily_cost = cluster_cost_hr * (R / 60) * J
Monthly waste	waste_month = daily_cost * waste * 30
Efficiency score	score = round(100 * (1 - waste))