Cloud Cost Refinery

Why This Matters

Cloud costs grow unchecked without active monitoring. Resources accumulate like sediment: an EC2 instance someone spun up for testing, a SageMaker notebook from an experiment three months ago, EBS volumes detached and forgotten, VPCs in regions nobody uses. Each one small, together they add up to real money.

The typical cost optimization advice is frustratingly vague: "right-size your instances," "use reserved capacity," "delete unused resources." Helpful in principle, useless in practice. Engineers do not have time to hunt through every region looking for zombie resources.

What if an AI agent could scan your entire AWS footprint, identify specific waste, and output ready-to-run commands with estimated savings? Not "consider reviewing your EC2 fleet" but "aws ec2 stop-instances --instance-ids i-0abc123 --region us-east-1 (saves $45/month, instance has been idle for 30 days)."

The "refinery" metaphor is intentional: we process raw billing data and resource inventory, extract the valuable signal, and eliminate the waste.

The Product Vision

The system connects to AWS via IAM role (read-only for safety). It pulls Cost Explorer data, CloudWatch metrics, and resource inventories across all regions. The AI analyzes utilization patterns, identifies waste, and generates an actionable report.

The output is not a list of suggestions. It is a list of commands:

```
IDLE SAGEMAKER NOTEBOOKS (Save $450/month)
- ml-experiment-jan: No activity 45 days
  aws sagemaker stop-notebook-instance --notebook-instance-name ml-experiment-jan --region us-east-1

UNATTACHED EBS VOLUMES (Save $240/month)
- vol-0a1b2c3d (500GB, us-east-1): Detached 60 days ago
  aws ec2 delete-volume --volume-id vol-0a1b2c3d --region us-east-1
```

Users can review, modify, and execute. Dry-run mode previews without acting. The system respects exclusion tags (DoNotDelete) and requires confirmation for anything destructive.

What It Finds

Idle compute: EC2 instances with CPU below 5% for 14+ days. SageMaker notebooks with no activity for 7+ days. Lambda functions that have not been invoked in 30+ days.

Orphaned storage: Unattached EBS volumes. Snapshots of deleted volumes. S3 buckets that are empty but incurring lifecycle/versioning costs.

Network waste: Elastic IPs not associated with anything. NAT Gateways processing zero bytes. VPCs in regions with no actual resources.

Over-provisioned resources: RDS instances with consistently low CPU and few connections. EC2 instances that could be downsized. EBS volumes with low IOPS utilization that could use cheaper storage classes.

Missed savings opportunities: On-demand instances that have run continuously for months (candidates for reserved/savings plans). Resources in regions with higher pricing when they could run elsewhere.

Agentic Architecture

Discovery Agent inventories all resources across all regions. This is more complex than it sounds: AWS has dozens of services, each with its own API for listing resources. The agent builds a comprehensive map of what exists.

Metrics Agent pulls CloudWatch data for utilization patterns. CPU, memory, network, disk I/O, invocation counts, connection counts. It handles the quirks of CloudWatch (delayed metrics, different granularities, missing data points).

Cost Agent pulls Cost Explorer and billing data. Maps costs to specific resources where possible (not always straightforward due to shared costs, data transfer, etc.).

Analysis Agent combines inventory, metrics, and costs to identify waste. This is where the AI reasoning happens: is this instance idle because it is unused, or because it is a batch job that runs once a week? Context matters.

Command Generator produces executable AWS CLI commands for each recommendation. Includes estimated savings, risk assessment, and rollback instructions where applicable.

Implementation Hints

Start with the easy wins. Unattached EBS volumes, unassociated Elastic IPs, and stopped-but-not-terminated instances are unambiguous waste. Get these working reliably before tackling nuanced cases like "is this instance really idle?"

Handle the multi-region sprawl. AWS has 30+ regions. Scanning all of them takes time and API calls. Implement parallel scanning with rate limiting to avoid throttling.

Be careful with shared resources. That "idle" EC2 instance might be a bastion host that is critical for emergency access. That "unused" VPC might be peered with production. Surface dependencies before recommending deletion.

Respect organizational boundaries. In multi-account setups, resources in one account may depend on resources in another. The system needs awareness of these relationships.

Build in safety layers. Dry-run by default. Require explicit confirmation for destructive actions. Generate rollback instructions (though some actions like deleting volumes are not reversible—make this clear).

Track what was recommended vs. what was done vs. what the impact was. This feedback loop is valuable both for improving the AI and for demonstrating ROI to stakeholders.

Interesting Questions

How do you handle shared resources across teams? The database that Team A owns but Teams B, C, and D also use. Whose cost is it? Who can decide to downsize it?

What is the approval workflow for production resources? Finding waste is easy. Getting permission to delete it is organizational, not technical. How does the tool integrate with existing change management?

Should recommendations be auto-applied? Some organizations might want "delete all unattached volumes older than 90 days" to run automatically. Others would never allow it. How do you support both?

Can the AI predict future costs? Based on current trends, usage patterns, and planned projects, what will next month's bill look like? This moves from reactive cleanup to proactive planning.

Should this support GCP and Azure? Multi-cloud is common. The patterns are similar but the APIs are completely different. Is it worth the complexity?

Getting Data for Validation

Unlike market data or Spark logs, **there is no publicly downloadable AWS billing dataset**. AWS billing data is inherently private. Here are realistic approaches:

Your Own AWS Account is the most practical path. Use your AWS Academy or personal free tier account. Deliberately create wasteful resources: spin up an EC2 instance and let it idle, attach an EBS volume then detach it, allocate an Elastic IP and leave it unassociated, create a NAT Gateway in an unused VPC. After a few days, you have real Cost Explorer data and CloudWatch metrics showing the exact patterns your AI needs to detect. Enable Cost and Usage Reports to S3 to get detailed billing data. Stay within free tier or budget $10-20 for realistic test data.

Sample CUR Data is provided in cloud-cost-data-sample.csv as a reference. This shows the actual structure of AWS Cost and Usage Report data with anonymized account IDs. If you need more realistic data, request access to actual CUR exports from an AWS Academy account. The data provided will be limited but sufficient to understand the schema and generate larger synthetic datasets for testing.

Synthetic Data Generation is essential for testing at scale. The [CUR schema](https://gist.github.com/14kw/a82dc2c572472d492a7a17c36f6f0151) has ~300 columns including:
- `line_item_usage_account_id`, `line_item_product_code`, `line_item_usage_type`
- `line_item_usage_amount`, `line_item_unblended_cost` (the key cost fields)
- `line_item_usage_start_date`, `line_item_usage_end_date`
- `product_instance_type`, `product_region`, `product_operating_system`

Write a generator that creates realistic billing records: 1000 EC2 instances with varying utilization (some idle at $0.10/hr doing nothing), unattached EBS volumes accruing storage costs, Elastic IPs charged $0.005/hr when unassociated. This lets you test edge cases and volume.

cloud-cost-analyzer ([ssp-data/cloud-cost-analyzer](https://github.com/ssp-data/cloud-cost-analyzer)) includes a demo mode with sample data. Run `make demo` and it opens at localhost:9009 with pre-loaded dashboards. Good for understanding what a cost analytics UI looks like and what data structures to target.

Reference Implementations show how experts encode detection rules:
- [Cloud Custodian](https://cloudcustodian.io/) has hundreds of policies for identifying waste (idle instances, unattached volumes, etc.)
- [OptScale](https://github.com/hystax/optscale) is a full FinOps platform with detection heuristics
- [AWS CUR Query Library](https://www.wellarchitectedlabs.com/cost-optimization/cur_queries/) provides SQL queries encoding domain expertise

Instance Type Reference Data from [aws-samples/aws-usage-queries](https://github.com/aws-samples/aws-usage-queries) includes CSV files with instance families, vCPU counts, and pricing tiers useful for rightsizing recommendations.

The practical path: generate synthetic CUR data based on the schema, validate detection logic against Cloud Custodian rules, then test against your real AWS account with deliberately wasteful resources.

The Bigger Picture

Cloud cost management is a growing discipline (FinOps) with its own certifications, conferences, and tools. This project is not about replacing those tools but about exploring how AI agents can make cost optimization more accessible.

The interesting research question is whether an AI can match the judgment of an experienced cloud architect who knows which resources are actually critical and which are cruft. That judgment comes from organizational context that the AI may not have access to. How do you bridge that gap?