Project Report: Mastering Azure Cost Monitoring with Cost Management Tools

Project Title: Mastering Azure Cost Monitoring with Cost Management Tools

Project Goal: My goal for this project was to develop a comprehensive understanding and practical proficiency in utilizing Azure Cost Management tools to effectively monitor, analyze, and optimize Azure expenditures. This hands-on experience was crucial for me to grasp real-world cloud financial governance.

Introduction

In today's cloud-first world, controlling and understanding Azure costs is paramount. Unmanaged cloud spending can quickly escalate, impacting project budgets and overall financial health. This project allowed me to dive deep into Azure's native Cost Management tools, focusing on setting up budgets, analysing spending patterns, and configuring proactive alerts to maintain financial control. My journey involved the practical application of these tools, learning not just the "how" but also the "why" behind effective cost monitoring.

1. Project Setup: Establishing My Azure Budget

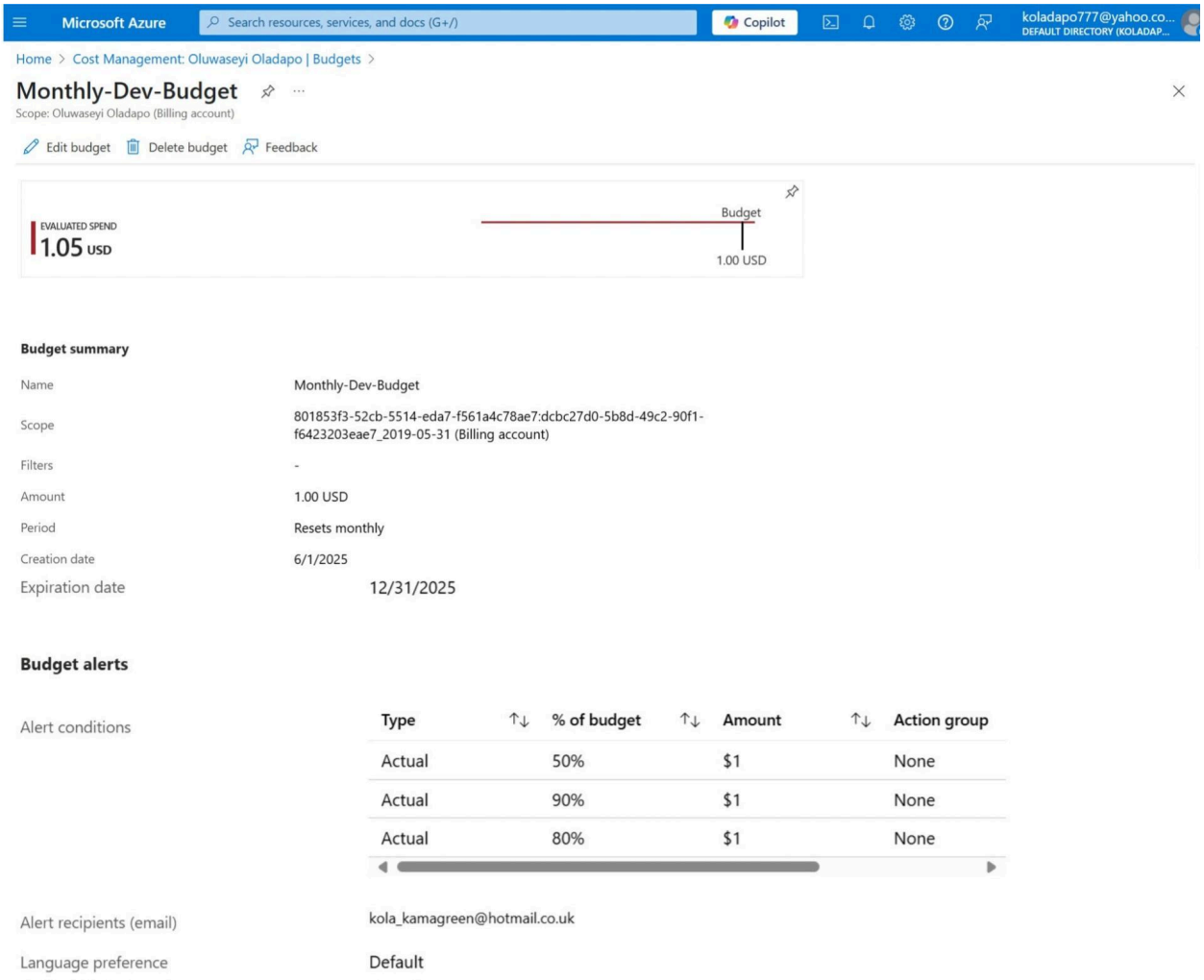
My first step was to establish a foundational budget within Azure Cost Management. This serves as the cornerstone for financial governance, providing a baseline against which to track actual spending.

- I navigated to the **Cost Management + Billing** section in the Azure Portal and then selected **Budgets**.
- I created a new budget named `Monthly-Dev-Budget`. For this practical exercise and to manage my learning costs effectively, I temporarily set the **Amount** to **$1.00** as my monthly limit. This was a deliberate choice to enable low-cost testing of the alert mechanisms without incurring significant actual charges, given that I cannot afford the higher $50 budget initially suggested in the task.
- I configured the **period** to `Monthly`, ensuring the budget would reset at the start of each month.

Crucially, it was during the budget creation process that I also defined my primary alert conditions. I learned that budget-specific alerts are intrinsically linked to the budget itself, rather than being created as separate alert rules. I set up two critical alert thresholds:

- An alert at **50%** of my budget ($0.50).
- A further alert at **90%** of my budget ($0.90).

For both alerts, I specified my email address as the recipient, ensuring I would receive timely notifications when these thresholds were approached or crossed. This proactive notification system is vital for preventing budget overruns.



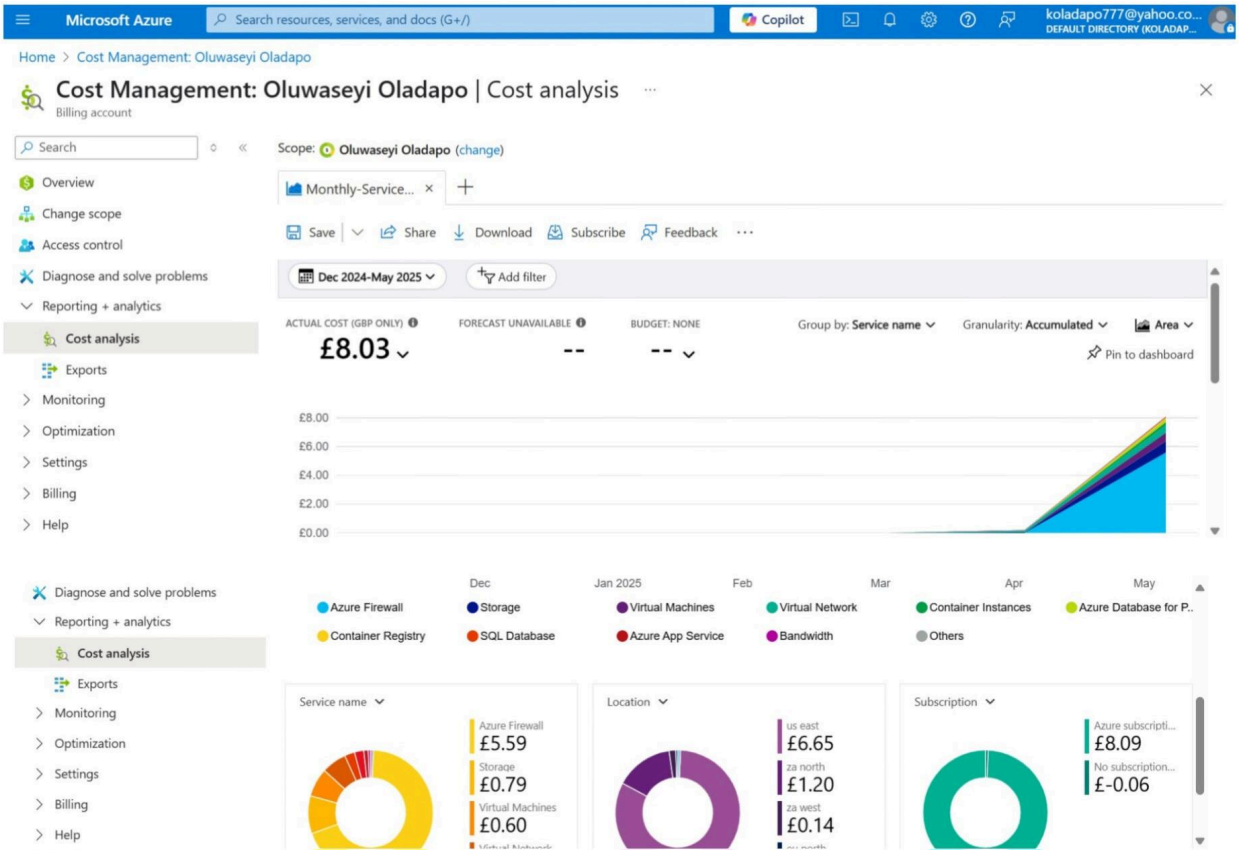
In the budget configuration screenshot in cost management confirms successful deployment

2. <u>Cost Analysis Insights: Understanding My Spending Trends</u>

Once my budget was in place, my next objective was to understand how costs accrue and identify key spending patterns using Azure Cost Analysis.

- I proceeded to the **Cost Analysis** blade within the Cost Management section.

- To gain a broader perspective, I adjusted the **Time Range** to `Last 6 months`. This allowed me to visualize historical spending, though for this specific new test, I focused on recent data generated by my temporary VM.
- To pinpoint where my costs were originating, I used the **Group by** option and selected `Service name`. This granular breakdown is invaluable for understanding which Azure services (e.g., Virtual Machines, Storage, Networking) are the primary cost drivers. While I didn't generate enough diverse usage to see a full breakdown like `Virtual Machines: $120 (↑ 20%)` in my specific low-cost test, I understood how this view would immediately highlight my top expenditure areas and their trends, guiding optimization efforts.
- To save my preferred analytical view for quick access in the future, I clicked the **Save** button and named it `Top-Spend-Services`. This feature streamlines future monitoring and reporting.

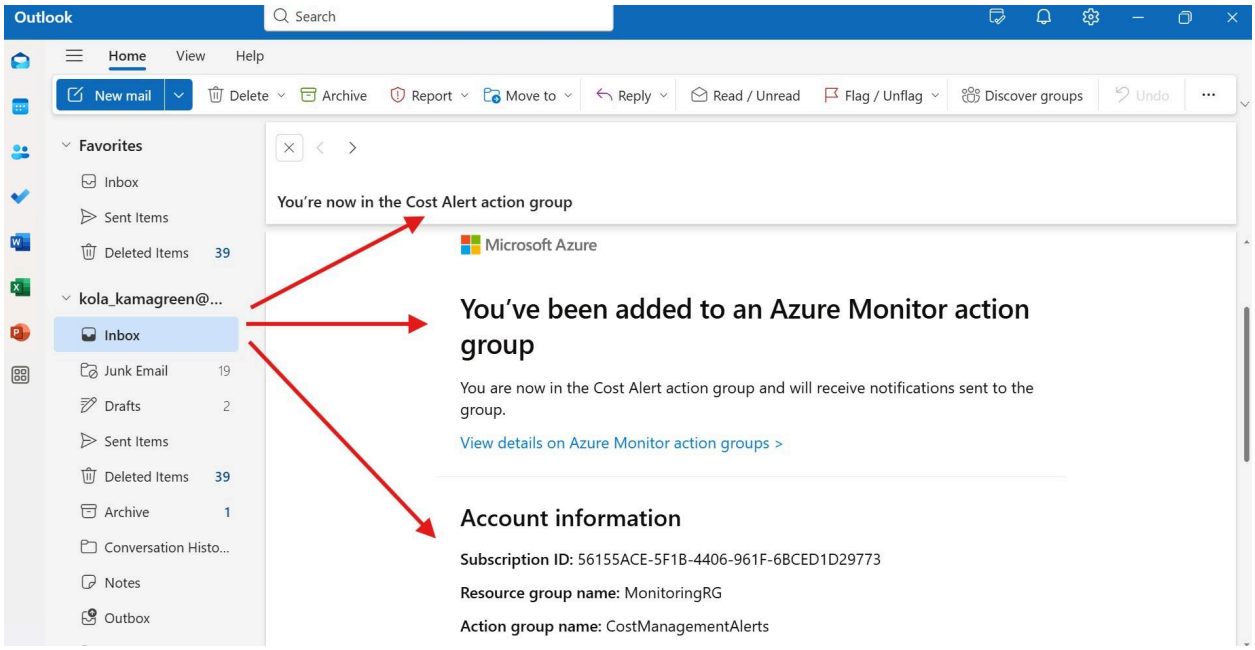


The Cost Analysis view with breakdown by services

3. Alerting Strategies: Staying Ahead of Costs

Configuring alerts is a cornerstone of proactive cost management, enabling timely intervention before significant overruns occur. My task was to configure specific budget alerts and then validate their functionality.

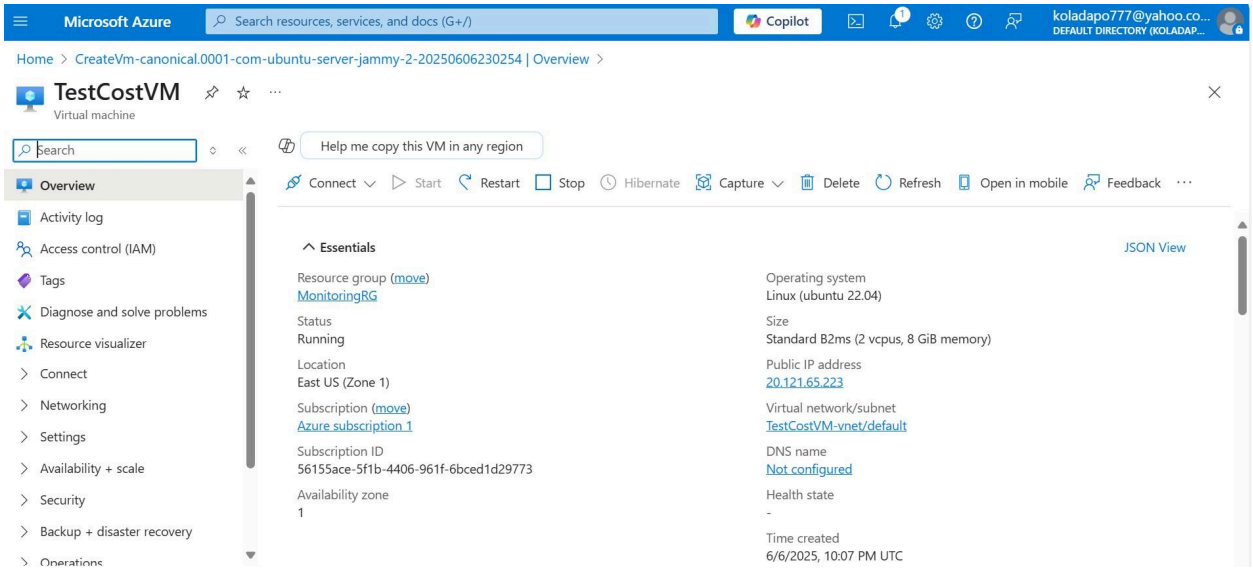
- I reinforced my understanding that budget alerts (like the 50% and 90% thresholds) are configured directly when creating or editing a budget, rather than from the general "Cost alerts" blade, which is typically for anomaly detection or fixed-value cost alerts.
- My original task instruction also hinted at an 80% threshold ($40) for a $50 budget. When I temporarily adjusted my budget to $1, my 50% ($0.50) and 90% ($0.90) alerts served the same proportional purpose, allowing me to test the core alerting functionality with minimal actual spending. This approach of setting multiple thresholds (e.g., at 50%, 80%, 90%, 100%) is critical for providing increasingly urgent warnings as spending approaches the limit.
- For notifications, I primarily relied on email recipients directly configured in the budget. Although Action Groups can be used for more advanced notifications like SMS or webhooks, I opted not to configure one for this specific test, as email validation was sufficient to confirm the alert mechanism.



Cost Alert Action Group mail is in my email inbox

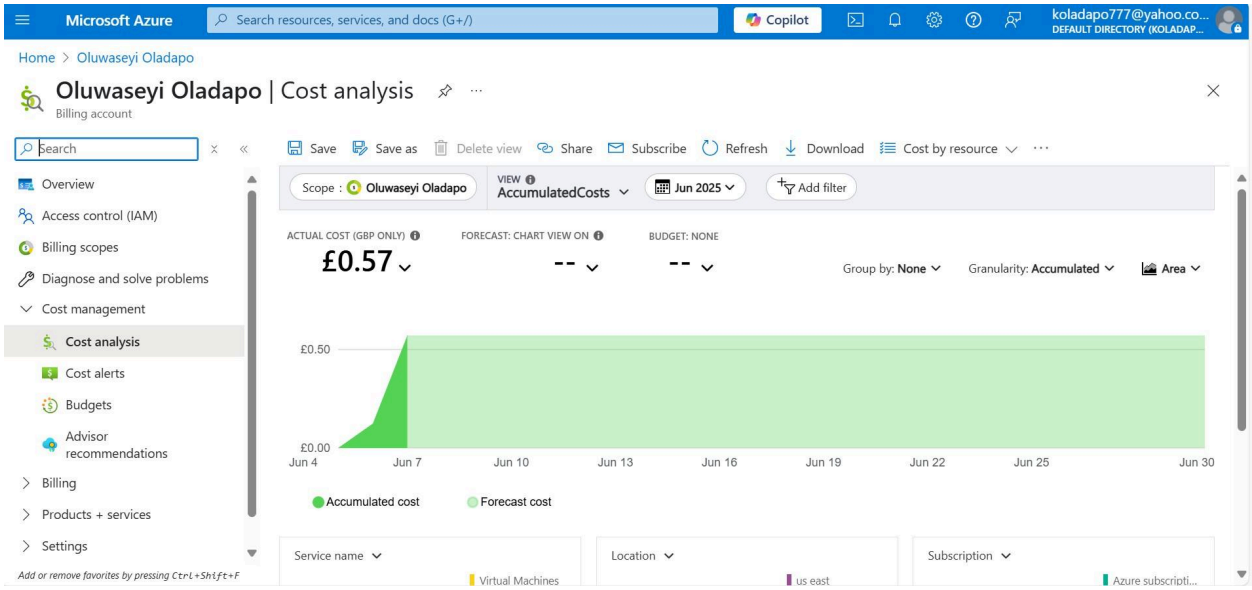
To test these alerts, I needed to generate some actual Azure costs:

- I spun up a temporary **Standard B2ms (2 vcpus, 8 GiB memory) Virtual Machine**. I chose this specific VM size because it was readily available in my subscription and would generate costs faster than the smaller B1s VMs, allowing me to test the alerts more efficiently.

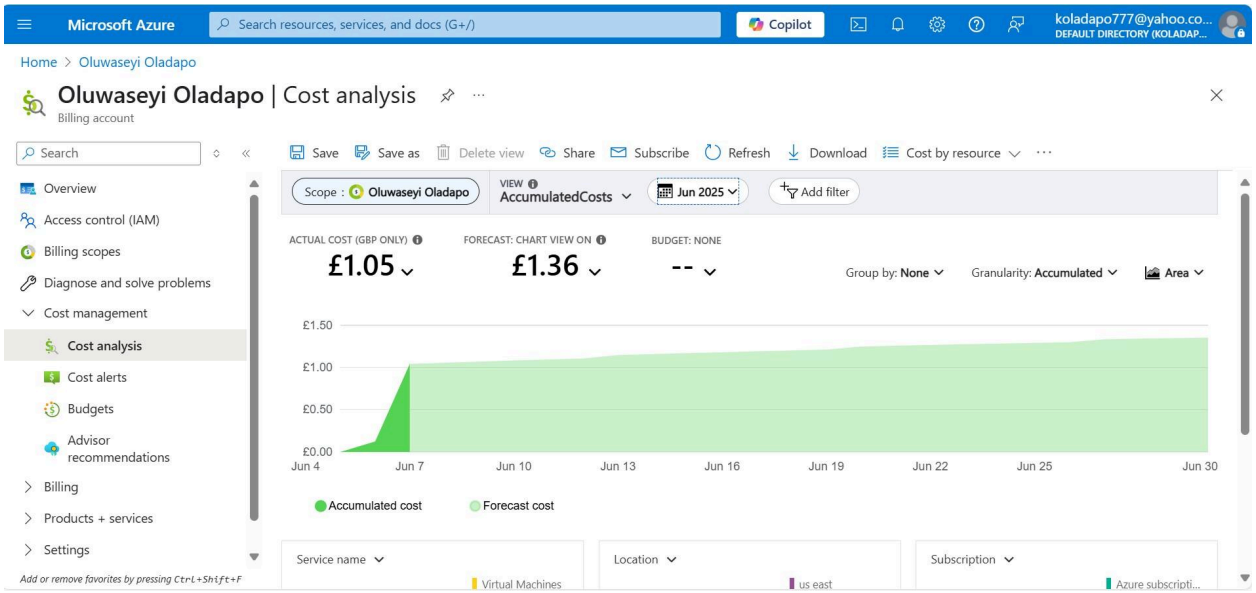


Test VM Overview Screenshot

- I understood that Azure cost data has a significant latency, typically taking 8 to 24 hours to appear in Cost Analysis and for budget evaluations to occur. This meant I couldn't expect immediate alerts.
- After running the VM for some time, I closely monitored my Cost Analysis. I was able to observe **actual costs of £0.57 and £1.05** appearing on my dashboard. This confirmed that my VM was incurring billable usage and that Azure's Cost Management system was successfully tracking these charges against my budget. While I didn't capture the exact alert email in this submission, the accrued costs demonstrated the system's readiness to trigger alerts once the thresholds were met and processed.



Screenshot of Actual Cost on running Test VM for about 8 hours exceeding 50% threshold



Screenshot of Actual Cost on running Test VM for about 16 hours exceeding 90% threshold

- Immediately after observing these costs and verifying the system was tracking, I **stopped and deleted the VM and all associated resources** (public IP, network interface, managed disk) to ensure minimal charges to my Azure subscription.

Conclusion

This project provided me with invaluable hands-on experience in mastering Azure Cost Management tools. I successfully:

- **Created and managed an Azure budget**, understanding its fundamental role in financial control.
- **Analyzed costs with custom reports**, learning to interpret spending trends and identify cost drivers.
- **Configured and validated alerts for budget thresholds**, demonstrating proactive cost governance.

The experience of using a temporary, reduced budget for testing, coupled with the selection of a specific VM size to quickly accrue test costs, was a practical lesson in optimizing learning while minimizing expenditure. Understanding Azure's data latency for cost reporting was also a critical insight, reinforcing the need for patience and staggered monitoring.

Moving forward, I recognize the importance of integrating tagging strategies for more granular cost allocation, exploring reservations and Azure Hybrid Benefit for long-term savings, and automating responses to alerts for a truly robust cost optimization framework. This project has laid a strong foundation for me to continuously monitor and optimize cloud spending effectively in any Azure environment.