

Summary Report: Securing Critical Resources with Azure Resource Locks

I'm pleased to share the findings from my recent project, "Securing Critical Resources with Azure Resource Locks." This project focused on understanding and implementing Azure's built-in capabilities to prevent accidental modification or deletion of crucial cloud resources.

My primary objectives for this project were clear:

- To truly grasp the distinctions between Azure's "Read-only" and "Delete" lock types.
- To practically apply these locks to a test resource, simulating a production environment.
- To rigorously validate that these locks enforce the intended protections.

Throughout the project, I worked with a test resource group, `ResourceLockRG`, and a virtual machine, `TestReadOnlyVM`, within it. I made sure my Azure account had the necessary Owner/Contributor permissions, which were essential for applying and testing the locks.

Here's what I found:

Understanding Azure Resource Lock Types

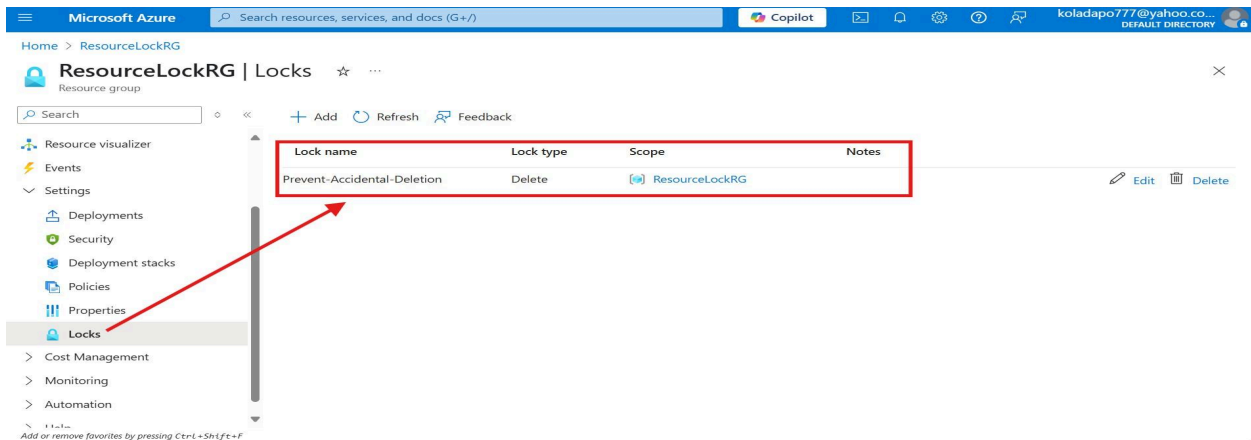
I learned that Azure offers two distinct types of management locks, each serving a critical purpose:

1. **Delete Lock:** This lock prevents any user from deleting the locked resource. While it blocks deletion, it still allows for modifications to the resource. My primary use case for this would be to safeguard critical production databases from accidental removal.
2. **Read-only Lock:** This is the most restrictive lock. It prevents both deletion and any modifications to the resource, allowing only read access. I immediately saw its value for scenarios requiring strict compliance or protecting static configurations, like network security group rules, where stability is paramount.

An important detail I confirmed is that these locks are inherited. If I apply a lock at the resource group level, it automatically extends its protection to all resources contained within that group.

Applying and Testing the Locks

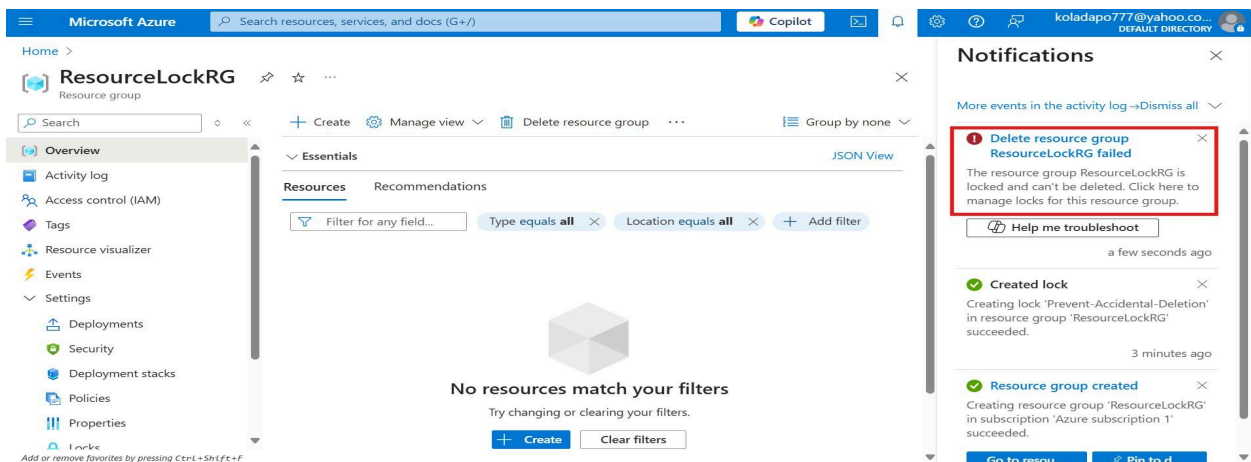
First, I applied a **Delete lock** to my **ResourceLockRG** resource group, naming it **Prevent-Accidental-Deletion**. This was where I began rigorously testing the lock's enforcement.



Screenshot of Delete Lock Applied to a Resource Group

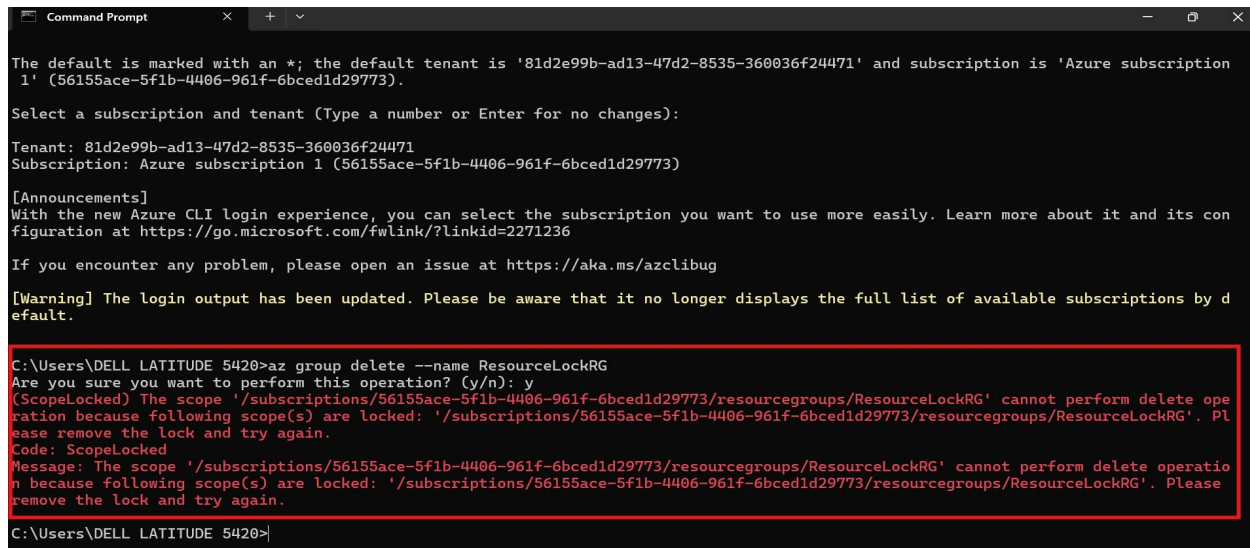
TEST 1: ATTEMPT DELETION (DELETE LOCK)

When I tried to delete the **ResourceLockRG** resource group directly from the **Azure Portal**, the deletion operation was blocked. I received an error message clearly stating that the resource could not be deleted due to a 'Delete' lock. This visually confirmed the lock's immediate effect within the graphical interface.



Screenshot of failed deletion attempt through the Azure Portal

Following that, I proceeded to test the deletion via the **Azure CLI**. I executed the **az group delete** command for **ResourceLockRG**, and just as in the portal, the operation failed. The command line output also explicitly indicated that the scope was locked and could not perform the delete operation. This verified that the lock's protection extends consistently across different management interfaces.



```
Command Prompt

The default is marked with an *; the default tenant is '81d2e99b-ad13-47d2-8535-360036f24471' and subscription is 'Azure subscription 1' (56155ace-5f1b-4406-961f-6bcd1d29773).

Select a subscription and tenant (Type a number or Enter for no changes):

Tenant: 81d2e99b-ad13-47d2-8535-360036f24471
Subscription: Azure subscription 1 (56155ace-5f1b-4406-961f-6bcd1d29773)

[Announcements]
With the new Azure CLI login experience, you can select the subscription you want to use more easily. Learn more about it and its configuration at https://go.microsoft.com/fwlink/?linkid=2271236

If you encounter any problem, please open an issue at https://aka.ms/azclibug

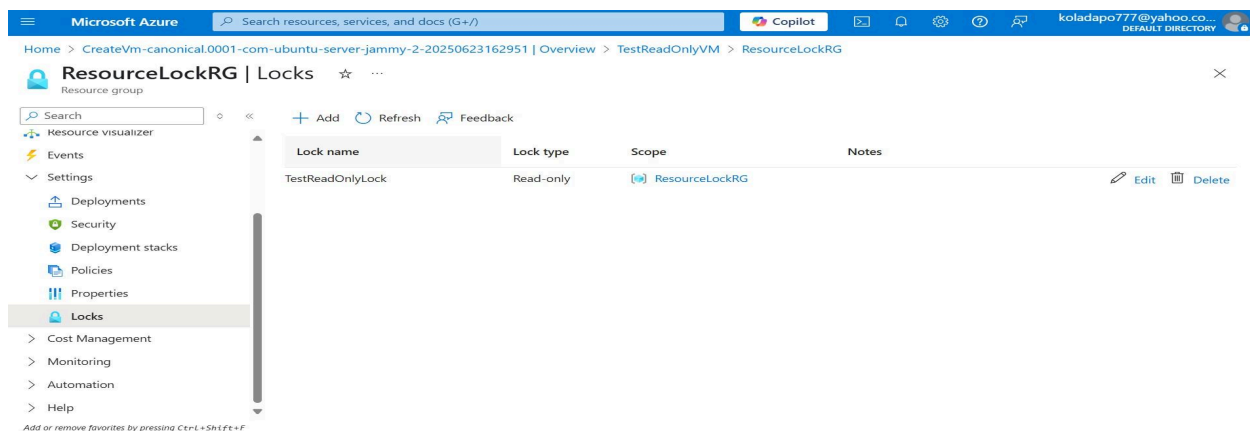
[Warning] The login output has been updated. Please be aware that it no longer displays the full list of available subscriptions by default.

C:\Users\DELL LATITUDE 5420>az group delete --name ResourceLockRG
Are you sure you want to perform this operation? (y/n): y
(ScopeLocked) The scope '/subscriptions/56155ace-5f1b-4406-961f-6bcd1d29773/resourcegroups/ResourceLockRG' cannot perform delete operation because following scope(s) are locked: '/subscriptions/56155ace-5f1b-4406-961f-6bcd1d29773/resourcegroups/ResourceLockRG'. Please remove the lock and try again.
Code: ScopeLocked
Message: The scope '/subscriptions/56155ace-5f1b-4406-961f-6bcd1d29773/resourcegroups/ResourceLockRG' cannot perform delete operation because following scope(s) are locked: '/subscriptions/56155ace-5f1b-4406-961f-6bcd1d29773/resourcegroups/ResourceLockRG'. Please remove the lock and try again.

C:\Users\DELL LATITUDE 5420>
```

Screenshot of failed deletion attempt through the Azure CLI

Next, to dig deeper, I changed the lock on **ResourceLockRG** to a **Read-only lock**. This is where the power of this feature truly shone.

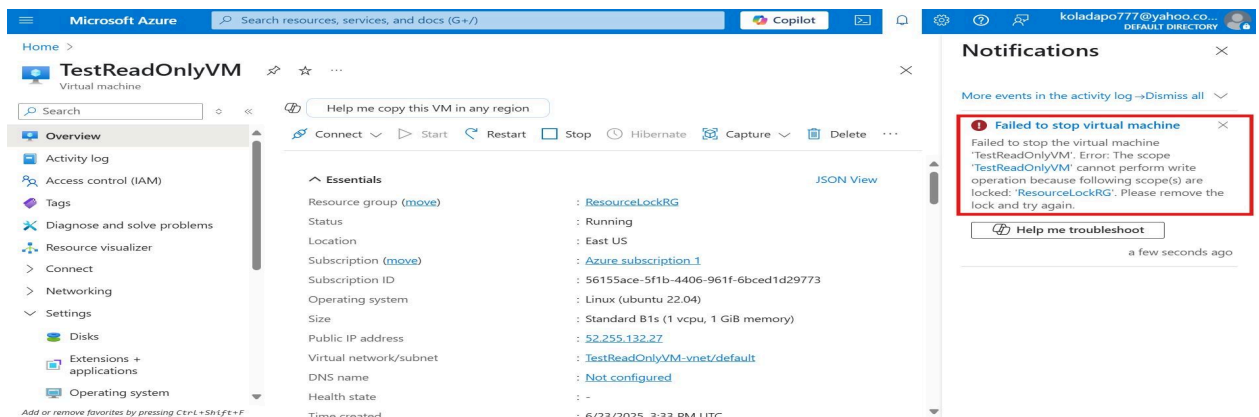


Screenshot of Read-Only Lock Applied to a Resource Group

TEST 2: VERIFY MODIFICATION (FOR READ-ONLY LOCKS)

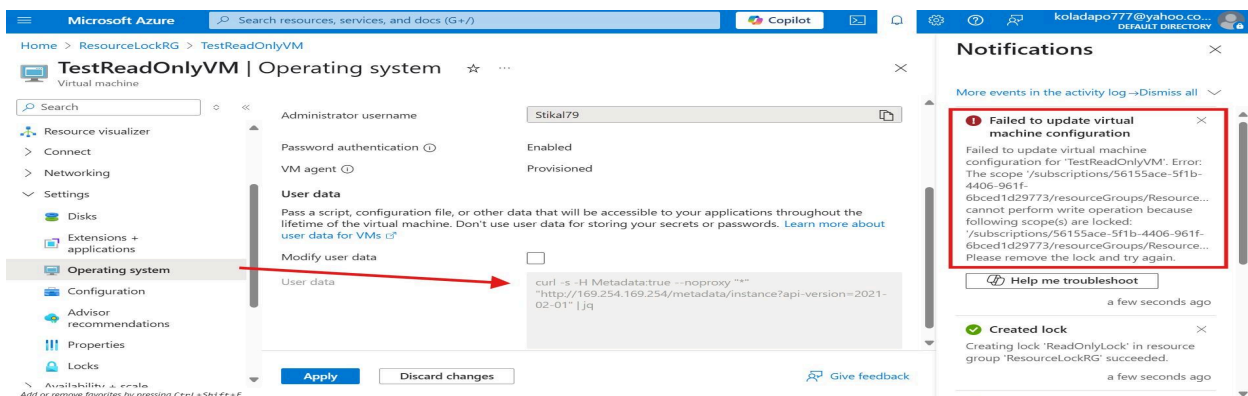
I then attempted to interact with the **TestReadOnlyVM** inside that locked resource group:

- **Attempting to Stop the VM:** When I tried to stop the **TestReadOnlyVM**, the operation was blocked. I received an error indicating that the VM inherited the 'Read-only' lock from its containing resource group. This immediately showed me that **Read-only** prevents operational changes.



Screenshot of Failed Modification Attempt to Stop VM

- **Modifying VM User Data:** To push the boundaries further, I attempted to modify the operating system's user data for **TestReadOnlyVM**. Again, the operation failed with an error, specifically due to the 'Read-only' lock. This confirmed that the lock extends to detailed configuration settings, preventing any unintended changes.



Screenshot of Failed Modification Attempt to Update VM

Deeper Insights and Real-World Applications

Beyond the technical implementation, I gained significant insights into the real-world value of these locks:

- **Defence in Depth:** Resource locks are a vital layer complementing RBAC, preventing even authorised users from accidental destructive actions.
- **Mitigating Human Error:** Their primary value lies in acting as a safety net against common accidental deletions or misconfigurations in dynamic cloud environments.
- **Enforcing Governance and Compliance:** They provide a technical mechanism to enforce immutability for auditing or regulatory compliance, ensuring data and configurations remain untampered.
- **Supporting DevOps & Automation:** Locks can protect production resources from manual, out-of-band changes, promoting a "hands-off" approach to highly stable environments.

Potential Use Cases:

- **Protecting Core Production Infrastructure:** Essential for databases (preventing deletion), critical VMs (preventing deletion/modification for stable configs), and foundational network components (preventing removal of VNets, subnets, load balancers).
- **Securing Shared Services:** Ensuring the integrity and availability of centralised logging/monitoring accounts and identity management resources.
- **Compliance and Audit Scenarios:** Guaranteeing immutability for archival storage or preventing accidental deletion of Key Vaults.
- **Disaster Recovery (DR) Components:** Securing DR infrastructure, like Site Recovery Vaults, to ensure readiness for failover.
- **Higher-Level Governance:** Applying locks at the subscription or management group level for broad policy enforcement.

Conclusion

My findings confirm the effectiveness of Azure Resource Locks. Both "Delete" and "Read-only" locks performed as expected, preventing unauthorised actions. Their inheritance model makes them powerful for broad protection. This project reinforced that Azure Resource Locks are a simple yet vital tool for maintaining cloud infrastructure integrity, combating human error, enforcing governance, and enhancing security. I'm confident in applying these concepts to secure production environments.

