WHITEPAPER

Quali | TORQUE

Ephemeral Environments

Unlocking FinOps Value with Automated Ephemeral Environments for Non-Production Workloads

AI generated Image

Unlocking FinOps Value with Automated Ephemeral Environments for Non-Production Workloads

Introduction

While looking for ways to streamline the delivery of non-production environments, an enterprise software vendor working with the Quali team discovered a cloud cost savings opportunity that amounted to millions of dollars every year.

However, even while understanding how those savings could affect gross margins, leadership at this organization didn't believe it was feasible. The overhead required to orchestrate and manage the lifecycle of these environments would be so significant that the organization couldn't take the project on.

Unfortunately, this situation is far too common. Even advanced cloud-native organizations leave millions of dollars of FinOps value on the table simply because they lack the toolset to optimize the operational side of their cloud infrastructure.

In this paper, we'll walk through this specific example and highlight best practices among the Quali community to understand:

- The financial impact of ephemeral environments for non-production workloads.
- Automation capabilities that have helped to transition these workloads to ephemeral environments
- Additional cost optimization and productivity benefits accessible for ephemeral environments

What are Ephemeral Environments?

Ephemeral environments entail the provisioning and termination of infrastructure to support any workload that does not need to be available permanently.

Commonly used for use cases such as software development, testing, demos, training, or POCs, ephemeral environments align infrastructure consumption with actual resource utilization.

What are Ephemeral Environments?

Ephemeral environments can consist of anything ranging from individual infrastructure resources, such as basic virtual machines, to complex definitions of dependencies between infrastructure, data services, applications, and anything else needed to deliver the environment output.

Ephemeral environments typically fall into one of two categories:

Scheduled environments: Commonly used for workloads that require on-demand access from large numbers of users at any point during standard business hours, these types of environments are provisioned at the beginning of the workday and then terminated at the end of the workday. This approach accounts for the on-demand nature of the infrastructure while eliminating unnecessary costs and runtimes during nights, weekends, and holidays.

Just-in-time environments: Ideal for short-term, standalone workloads, just-in-time environments further streamline cloud costs by aligning the provisioning and termination of these resources with the actual duration in which they're needed. Examples include short-term testing phases in software development pipelines.

Determining which approach is best requires understanding how your teams use these environments, how frequently they'll need them, and who will be responsible for ensuring their availability. Naturally, a just-in-time approach will entail lower costs than those that run the full business day but could lead to redundant provisioning processes and access delays if users need them at various points throughout the day.

When deciding on the right approach to implementing ephemeral environments, it's critical to evaluate the impact on productivity, staffing, and user experience in addition to costs on the cloud bill.

Comparing the Types of Ephemeral Environments

Scheduled Environments	Just-in-Time Environments
• Set to run only during standard business hours	• Set to run only for the duration of the workload it supports
• Access needed at various points throughout the day	• Access only needed for short-term, standalone workloads
• Eliminates redundant provisioning every time they're needed	• Eliminates redundant provisioning every time they're needed
• Prevents wasted cloud costs during nights, weekends, and holidays	• Prevents wasted costs due to unnecessary runtimes when the workload isn't needed

Background: A Case Study in Wasted Cloud Costs

This paper will examine an enterprise software vendor that maintains a unique production environment supporting each of its customers' workloads.

In support of each of these customer-specific production environments, the software vendor also maintains several cloud-based environments for non-production tasks, such as development and testing.

Financially, the cost to deliver these non-production workloads would hold back profitability continually. The onboarding of new customers, as well as the delivery of new features in support of those customers, entailed the creation and maintenance of not only a unique production environment for each customer, but also the non-production environments supporting them.

Subsequently, revenue growth requires increased cloud costs to deliver these environments, eating up profit margins as a result. Any opportunity to trim the cloud costs for these environments would have a direct impact on profitability.

These non-production workloads were perfect candidates for ephemeral environments, or those which run only during the hours in which they're needed. By aligning runtimes with the team's actual needs, converting non-production workloads to ephemeral environments could eliminate cloud costs without affecting day-to-day operations at all.

However, even while recognizing the potential impact of ephemeral environments on operational expenses and gross profit margins, this organization still faced significant barriers to adopting some of the measures that would unlock this value.

Any plan to implement this change would need to eliminate complexity and employ automation to integrate with the engineering team's existing workflows and capacity.

1 Calculating the Financial Benefits of Ephemeral Environments for Non-Production Workloads

One critical consideration for this project was to identify which non-production workloads were eligible to be converted to ephemeral environments. For example, some environments for backup and testing were required for 24/7 availability to comply with customer agreements.

After reviewing all non-production workloads and accounting for these types of requirements, the Quali team identified more than 350 environments that were running 24/7 but were only needed for activity during standard business hours

This amounted to about $1.9 million in annual cloud costs for non-production workloads alone.

Some basic math uncovered that more than 50% of runtimes for these non-production workloads were unnecessary. Simply eliminating runtimes overnight, on weekends, and on holidays, would eliminate a substantial amount of cloud costs without affecting day-to-day productivity.

Eliminating these unnecessary runtimes would amount to an estimated $1.26 million in annual cloud cost savings.

Cost Savings Calculation

Total Hours of Operation	**8,760 Hours**
Idle Hours Identified	**4,4760 Hours** (54% of total)
Weekends & Holidays	**2,760 Hours** 24 hours × 115 days
Workday Off-Hours	**2,000 Hours** 8 hours × 250 days
Active Hours Required	**4,000 Hours** (46% of total)

2 Requirements for Implementing Ephemeral Environments

Uncovering the cost-savings opportunity was not the primary challenge. Obviously, consuming fewer cloud services will result in a lower cloud bill.

The true obstacle is acting on this waste.

Without the right tech stack, converting non-production workloads to ephemeral environments requires significant overhead. Accomplishing this transition requires:

Codifying Infrastructure:

Ephemeral environments require daily provisioning and termination of cloud resources, which would entail substantial manual work if approached through a manual ClickOps approach. Infrastructure as Code is critical to establishing this automation.

Codifying Environments:

Even with cloud services defined as code, many environments require configuring dependencies and input parameters for various components—infrastructure, applications, data services etc. This adds more overhead to the daily delivery of ephemeral environments, as few organizations can codify these environments so they can be managed similarly to Infrastructure as Code.

Cloud Governance:

As the volume of reusable assets for infrastructure and environments grows, so does the risk of security vulnerabilities and inflated cloud costs due to misconfigured and over-sized resources.

Lifecycle Management:

Aligning cloud operations with utilization requires the timely provisioning and termination of infrastructure, which can be hampered by human error, failures in IaC commands, and drift due to staff turnover or other organizational changes over time.

Maintaining Infrastructure:

Updates to infrastructure code or routine activity like security patches and application upgrades can result in unexpected side effects on infrastructure, which could include reliability of non-production environments as well as the effectiveness of lifecycle management.
If the implementation of ephemeral environments affects reliability or fails to align with standard processes for Day 2 activity, the engineering team is likely to abandon the plan altogether.

Monitoring for Reliability & Performance:

Similarly, the ability to diagnose and reconcile unexpected errors, configuration drift, or other updates will need to align with the operation of ephemeral environments.

Implementation needs to account for these challenges while minimizing the impact on staff bandwidth.

3 Implementing Ephemeral Environments

To help navigate the challenges and ease the implementation and operation of ephemeral environments, this project employed generative AI, automation, and cloud governance via Quali's Torque platform.

Here is how it works:

Automating the Codification of Cloud Infrastructure

Without the right tech stack, converting non-production workloads to ephemeral environments requires significant overhead. Accomplishing this transition requires:

- Connecting to the user's public cloud accounts (with support for AWS and Microsoft Azure)
- Discovering the resources deployed via those accounts
- Automatically generating open-source Terraform IaC files defining the state of the resources discovered

Once codified, these IaC files can be downloaded and added to a repository. This can accelerate the creation of IaC files that will be used to create ephemeral environments without requiring extensive training or new staff with expertise in IaC.

Automating the Codification of Cloud Environments

Even after automating the creation of IaC files, the orchestration of environments threatens to derail the seamless integration of ephemeral environments.

The solution is to define Environments as Code, including the parameters for individual cloud resources and dependencies among them, so users can run and maintain environments continuously.

To accomplish this, Torque:

- Connects to the user's repositories (with support for GitHub, GitLab, Bitbucket, and others)
- Discovers and inspects the resource configurations defined in the IaC modules and other resources in those repositories, including those generated by Torque and otherwise (with support for Terraform, OpenTofu, Ansible, Helm, CloudFormation, and others)
- Normalizes these resources to eliminate differences in languages and other details so they can be used interchangeably as components in an Environment as Code blueprint
- Provides an AI Copilot for users to submit natural-language prompts describing how these resources should be configured to deliver an environment
- Designs the environment in Torque's interactive graphical UI showing dependencies and inputs for each resource, with the ability to modify the environment by clicking and dragging the components in the environment design
- Creates a file containing the code in line with the environment design, which automatically updates in response to changes made in the graphical UI tool

Once created, Torque can execute the code in these files—referred to as "blueprints" in Torque—to deploy environments repeatedly. These blueprints provide the foundation for ephemeral environments.

Automating the Lifecycle of Cloud Environments

While the inventory of reusable IaC modules and Environment as Code blueprints eliminates the manual codification of resources needed to support non-production workloads, they don't inherently address the overhead required to align cloud resource consumption with development work.

Torque accomplishes this with Workflows, which create code to define routine actions, such as the provisioning and termination of infrastructure. Workflows can be configured with cron expressions to automate these actions on a recurring basis.

These Workflows can automate the day-to-day operation of ephemeral environments in line with the workday off-hours identified in the calculation section of this paper, thereby providing uptime when non-production workloads are needed and eliminating wasted cloud costs on nights and weekends.

What are Environments as Code?

Environments as Code is an approach that defines all the components, and dependencies among those components, needed to deliver a workload as executable code in single file, referred to as a "blueprint."

This approach eliminates the redundant orchestration of environments, enables the seamless integration of environments into GitOps workflows and CI/CD pipelines, and simplifies the maintenance of infrastructure and other environment components by providing a single definition to manage.

Democratizing Secure, Self-Service Access to Environments

When managed with traditional tools for provisioning and managing Infrastructure as Code, access to environments often requires a ticket request that can only be fulfilled by a select few DevOps engineers or IT admins.

Security considerations often require this centralization of cloud access. To limit the distribution of cloud account credentials and other secrets needed to provision infrastructure, DevOps and IT teams are often the only ones empowered to deploy these resources. This contributes to DevOps and IT becoming a bottleneck.

To address this challenge, Torque:

- Provides all end users access to environment outputs directly via the platform
- Encrypts all cloud secrets and removes them from the provisioning process for those with end-user permissions, eliminating the need to distribute sensitive information to enable access to cloud resources
- Integrates with CI/CD platforms, Internal Developer Portals such as Spotify Backstage, and makes environments available via IDEs, CLIs, and other dev tools so they become a seamless part of the development lifecycle
- Manages custom role-based access controls to ensure that end users cannot modify environment configurations or deploy any new resources

To put this in the context of our earlier example, this approach provides software developers and other teams access to non-production environments they need—including with integrations into their operational tools—while ensuring they don't run overnight, on weekends, or on holidays.

4 Additional Cost Optimization Opportunities for Ephemeral Environments

The process of transitioning non-production workloads to ephemeral environments provides the monitoring, automation, and governance needed to further identify and prevent wasted cloud costs.

Here are a few additional ways to optimize cloud costs for non-production workloads:

Identifying Idle Resources During Work Hours

Idle cloud resources do not only occur outside of working hours. Operating ephemeral environments as code via Torque provides continuous monitoring for the capacity used among every resource deployed.

Based on this monitoring, Torque flags any live resource that is not using capacity as potentially inactive. The Inactivity report shows all such resources, including a calculation of wasted cost and the ability to view the user who deployed it, so engineering teams can communicate and make informed decisions.

This kind of visibility could lead to long-term decisions on which resources are considered critical and need to be deployed automatically on a daily basis.

Cloud Instance Right-Sizing Policy Enforcement

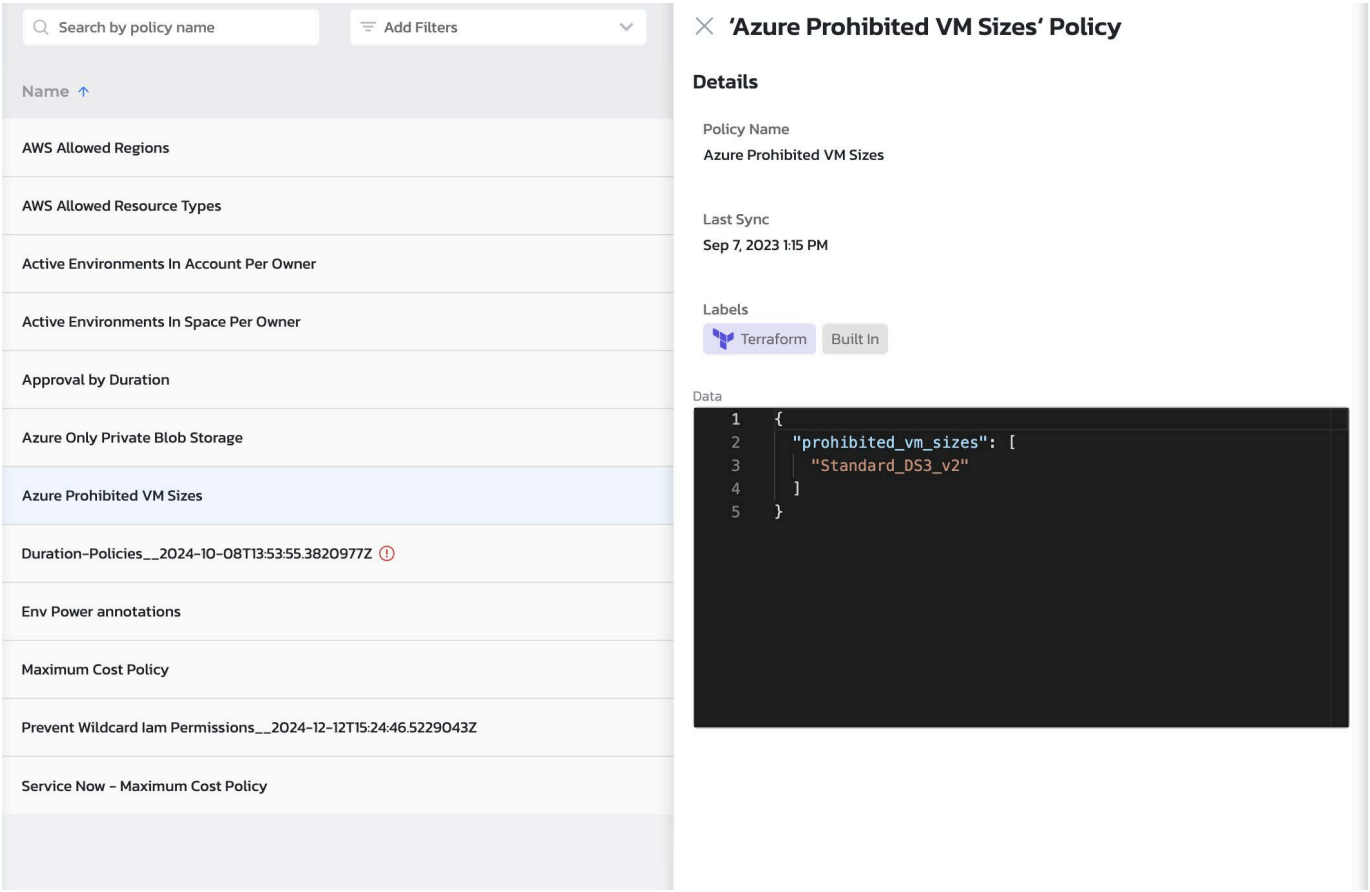
Right-sizing—or aligning the size of an instance to the actual needs of its workload—can have a significant impact on cloud cost optimization.

Implementing right-sizing, however, is often easier said than done. If engineers provision instances that violate their team's instance sizing guidelines, they likely won't know until after receiving the cloud bill. Similar to the management of the infrastructure lifecycle, any manual effort creates the risk of human error or drift following staff turnover.

Since Torque initiates the provisioning of all cloud instances, the platform can automatically deny activity that violates custom policies set by administrators.

This can include policies about instance sizes. If a certain team only requires certain sizes of instances, a policy instructs Torque to deny the action and can be set to trigger an approval workflow.

This proactively prevents the deployment of over-sized infrastructure, while also providing administrators the option to approve exceptions to the rule.



Maximum Cost Policy Enforcement

Similar to policies around instance sizes, many Torque users implement a policy for maximum expected cost.

Torque calculates the expected cost based on the resources in the Environment as Code blueprint (which can include individual IaC resources) and the environment duration set prior to deployment (which can be required to launch a resource).

Based on this calculation, Torque can deny the deployment and require approval if the expected cost exceeds the maximum expected cost policy.

This accelerates productivity by allowing engineers to run infrastructure that will not significantly increase cloud costs, while providing assurance that anything expensive was reviewed and approved by an administrator before it was launched.

Collaboration to Eliminate Redundant Infrastructure Deployments

Redundant deployments are an often-overlooked contributor to wasted cloud costs.

If, for example, five engineers provision identical cloud resources concurrently, the total cloud cost is five times more expensive than is necessary.

Torque users can share access to environments among individual collaborators, teams, or groups within a team. Collaborators are notified when an environment is live and provided a direct link to access the environment outputs.

To help promote this collaboration, Torque displays the number of active instances of each environment prior to deployment via the self-service catalog so end users are made aware that the environment they're trying to access is already live. If a user needs an environment but was not added as a collaborator, they can access the output on the Operation Hub, which shows all recently deployed environments for that team.

Another source of this waste is in the redundant deployment of infrastructure services supporting separate workloads.

Say, for example, two engineers need to run the same Kubernetes cluster to support different environments. In Torque, engineers with administrator permissions can share access to this Kubernetes cluster so it can be used as an input for separate environments. When those users provision their environments, they'll find that cluster available as an input—thereby eliminating the need to spin up that cluster for any subsequent use cases.

On a day-to-day basis, eliminating these redundant resources can add up to significant cloud cost savings.

Conclusion

The adoption of ephemeral environments is reflective of the evolution of cloud infrastructure management.

For years, DevOps and other engineering teams have prioritized speed via Infrastructure as Code and CI/CD, sacrificing control and governance as a result.

Building upon these principles, this approach supports long-standing DevOps practices for managing infrastructure while automating the kinds of tasks needed to optimize cloud costs.

To learn more, visit http://quali.com