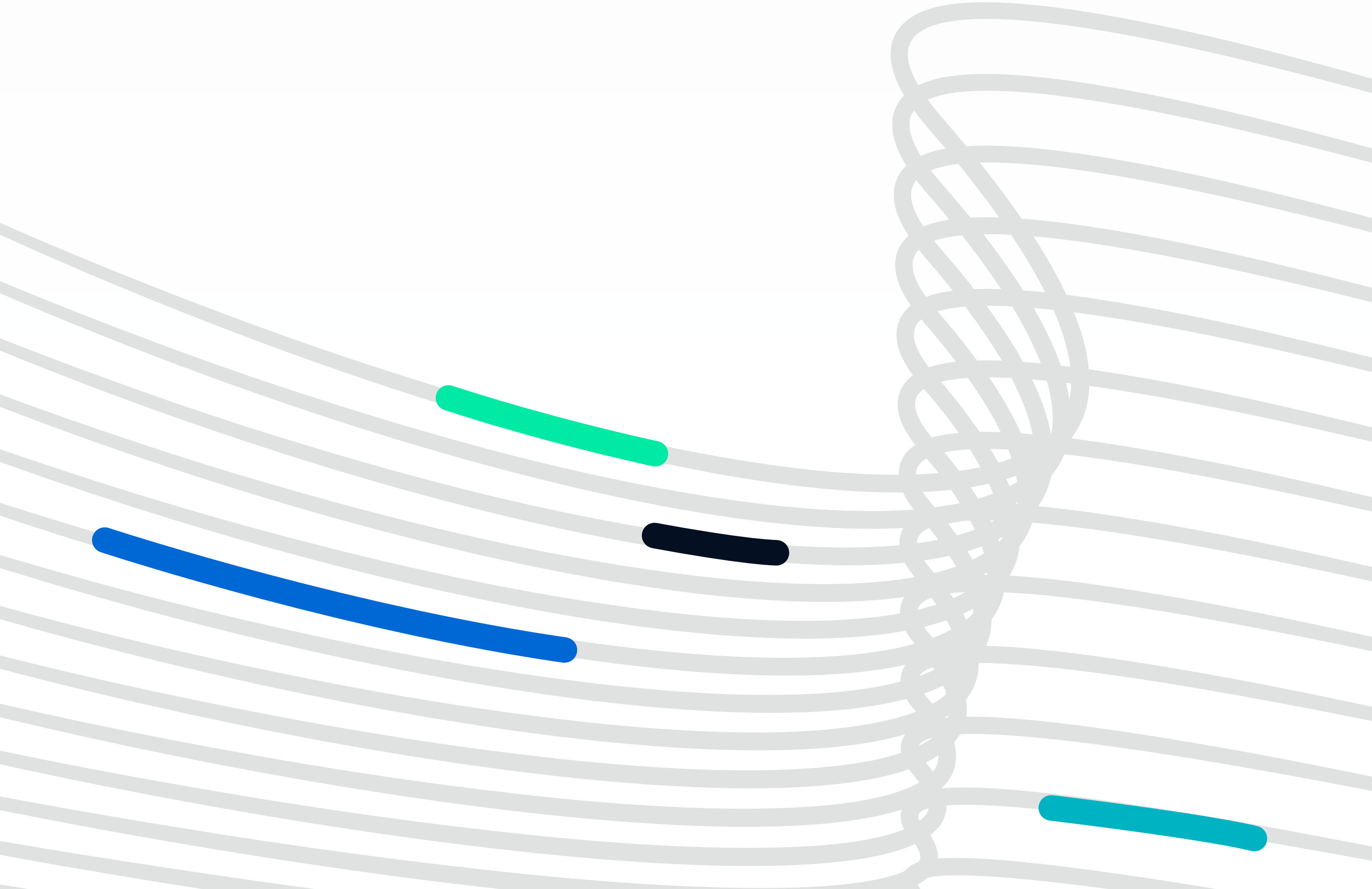
New Relic®

White Paper

Modernizing with AWS:

Getting Started with
'Observability-Driven' Refactoring

Part 3 of our 3-part series on legacy app modernization in the cloud



Introduction

As your organization moves existing applications to Amazon Web Services (AWS), you'll need to decide on the right approach for each application: rehosting, replatforming, or refactoring. This third white paper in our Application Modernization series covers the latter approach.

Refactoring in its simplest form means changing application code for the better. Typically, the goal is to improve performance, quality, and maintainability. Refactoring for AWS expands those benefits and enables additional ones such as better availability, scalability, and reliability. Another major benefit that refactoring provides is the ability to offload large amounts of undifferentiated work to AWS. Tasks such as hardware maintenance, operating system patch management, database updates, and more are all managed for you when using fully managed services such as Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), or AWS Fargate, allowing your team to focus on what differentiates your business.

Granted, refactoring is the riskiest of the modernization strategies, because it usually involves changing how an application works, either through logic or a significant shift in the underlying technology. At the same time, no other modernization approach delivers the level of positive impact and return on effort that refactoring can.

Read on to learn about the risks and benefits of refactoring, different types of refactoring in the AWS environment, and how New Relic can help you drive successful outcomes in the process.

What Is Refactoring?

Martin Fowler, author of Refactoring: Improving the Design of Existing Code, defines refactoring as "a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior."

Why Choose to Refactor?

Some of your applications are more critical to the business than others. Is an application strategically essential? Does it contribute revenue? Does it warrant further investment? If the answer is "yes" to any of these questions, it may be a candidate for refactoring.

While less strategic applications are more likely to be candidates for rehosting or replatforming, your strategic applications are the ones that are more likely to be your best choices for refactoring, because the investment you make will pay off in tangible outcomes. Some of the compelling reasons that companies choose to refactor an application include:

- Drive new revenue streams and/or optimize existing ones
- Create improvements that directly impact future revenue capabilities
- Deliver a better customer experience
- Enable faster time-to-market with new features
- Support a changing/new business model
- Comply with changing/new regulations
- Easily scale both up and down to meet planned and unplanned changes in traffic

Managing the Risk/Reward Balance of Refactoring

To make refactoring worth the risk and effort, it’s important to minimize the risk side of the equation while maximizing the reward side.

To do this, you need to understand what risks your organization might face when refactoring an application to/on AWS:

- Failure to fully understand the dependencies of the application, externally and internally
- Insufficient analysis to fully understand project complexity, resulting in failure to meet project goals
- Extensive changes that require extended testing and debugging with dual infrastructures, temporarily driving up costs
- Failure to achieve expected business outcomes
- Negative impact on the customer experience
- Unexpected introduction of new issues that need to be fixed, extending project length and cost

What many of these risks have in common is a lack of understanding of—and visibility into—the inner workings of the application. Data and observability significantly lessen the risks associated with the refactoring effort by helping you develop a deep understanding of the application before deciding how and what to refactor. For instance, using service maps in New Relic One, you can identify all of an application’s dependencies and take them into account when planning a refactoring effort.

The Essential Role of DevOps in Refactoring Success

Successful refactoring requires a strong culture of agility and DevOps. If you cannot easily test your application in minutes, then you won’t be able to regularly improve it. You can learn more about developing a robust, data-driven DevOps strategy, culture, and practices at newrelic.com/devops.

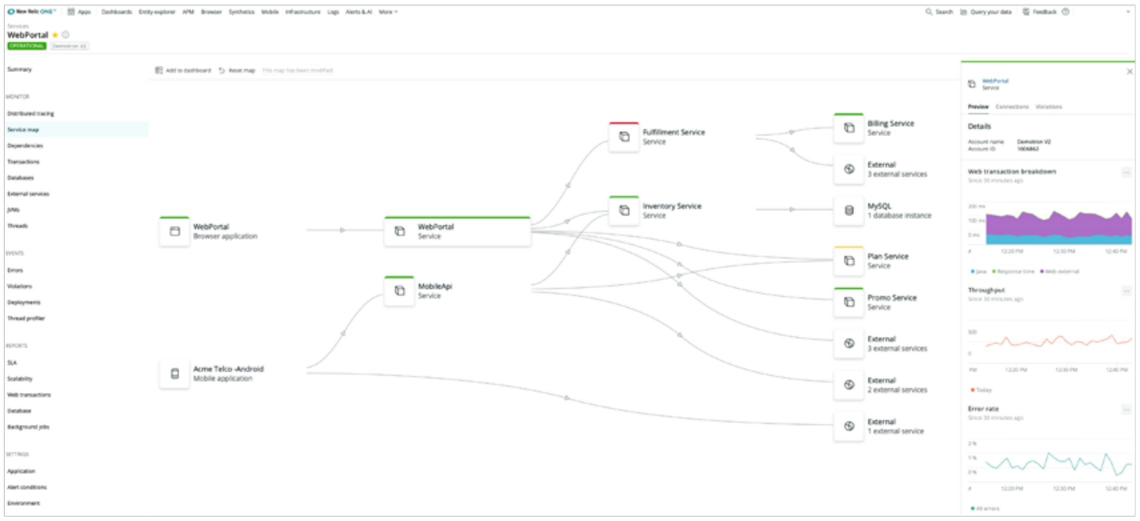


Figure 1. Explore application dependencies and components with the New Relic One service map.

The Value of Observability for Refactoring

The more clearly you understand an application as it exists today, the better the outcome of your refactoring effort. With observability of the application via monitoring and measuring, the insight about how the application is working in its current state extends throughout and beyond the immediate refactoring project to let you track and demonstrate the impact of your efforts.

The core principle of observability for refactoring is to instrument and measure everything before, during, and after the application has been refactored.

Before: To properly identify candidates, use New Relic One to gain a holistic view of the entire application, which gives you visibility and understanding into more than just the running state. Then you can correlate the running state with the codebase to understand where the biggest impact will be. Questions to ask of your data include:

- What parts of the codebase change the most or have the most issues filed against them? These are potentially good candidates for making them a component (i.e., a microservice running in a container or AWS Lambda).
- Which parts of the codebase have issues regularly reopened (which indicates rework)?
- Where are you doing significant amounts of work in code that could be handled by a component?
- Which parts of the application perform well? Which parts are complex and prone to errors? Which parts take the longest to run?
- Where could you use more modern technology to:
    - Mitigate performance problems?

   - Use serverless technologies?
   - Use self-healing infrastructure and services rather than manage the infrastructure?
   - Stop self-managing infrastructure if it does not give you a competitive advantage?
- How well are deployments going? How fast are things provisioned?
- What are the key performance indicators (KPIs) that help you measure business impact?

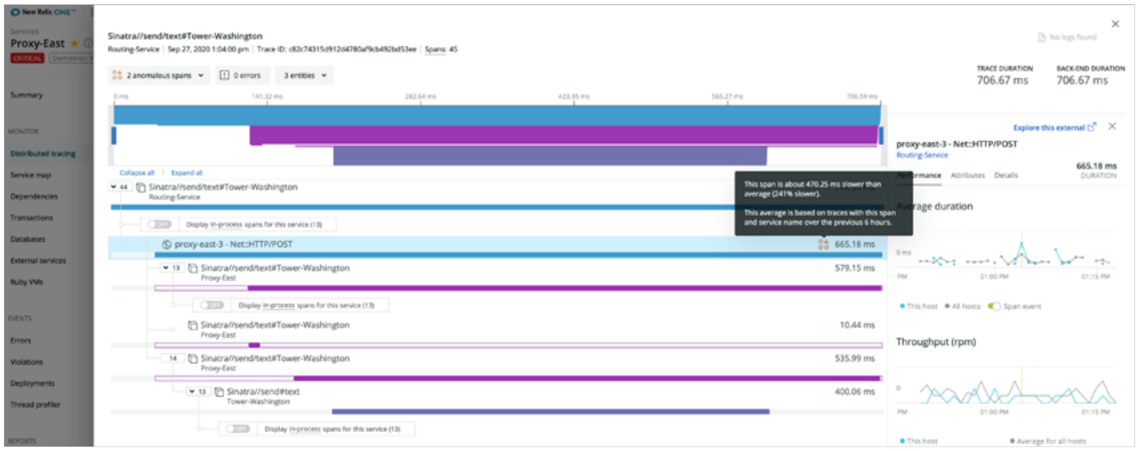

New Relic ONE
Proxy-East
Sinatra//send/text#Tower-Washington
Routing-Service | Sep 27, 2020 1:04:00 pm | Trace ID: c82c74315d912d4780af9cb492bd53ee | Spans: 45
2 anomalous spans
0 errors
3 entities
TRACE DURATION 706.67 ms
BACK-END DURATION 706.67 ms
proxy-east-3 - Net::HTTP/POST
665.18 ms
This span is about 470.25 ms slower than average (241% slower).
This average is based on traces with this span and service name over the previous 6 hours.
Average duration
Throughput (rpm)
Summary
Distributed tracing
Service map
Dependencies
Transactions
Databases
External services
Ruby VMs
Errors
Violations
Deployments
Thread profiler

Figure 2. Distributed tracing lets you see the path that a request takes as it travels through a distributed system. Use it to discover the latency of components along a path or to understand which component is creating a bottleneck.

During: Test the application to make sure refactoring isn’t causing problems. Use New Relic to monitor the performance of code and external services to understand how customers are being impacted during the switch. Determine how long you need to run dual infrastructures to ensure you have fully tested and switched over the refactored application(s).

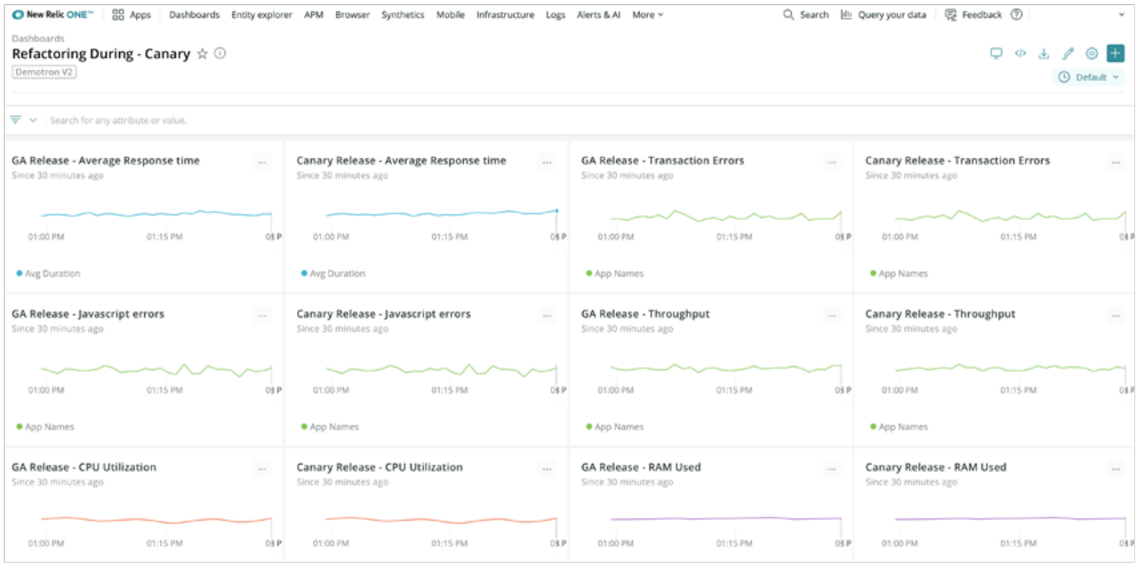

New Relic ONE
Apps
Dashboards
Entity explorer
APM
Browser
Synthetics
Mobile
Infrastructure
Logs
Alerts & AI
More
Search
Query your data
Feedback
Dashboards
Refactoring During - Canary
Demotron V2
Default
Search for any attribute or value.
GA Release - Average Response time
Canary Release - Average Response time
GA Release - Transaction Errors
Canary Release - Transaction Errors
GA Release - Javascript errors
Canary Release - Javascript errors
GA Release - Throughput
Canary Release - Throughput
GA Release - CPU Utilization
Canary Release - CPU Utilization
GA Release - RAM Used
Canary Release - RAM Used
Since 30 minutes ago
Avg Duration
App Names

Figure 3. Closely observe DevOps KPIs during refactoring.

After: In addition to understanding how refactoring affected application performance, you can also track other outcomes, including:

- **Issues:** Are fewer tickets being filed?
- **Costs:** Is this application costing you less in the cloud and/or is the current spend justified?
- **Technical debt:** Is the amount of rework less?
- **DevOps:** Are you deploying more often each day than you were previously? How much more?
- **Customers:** How was the customer experience impacted?
- **Business:** How were business KPIs impacted?

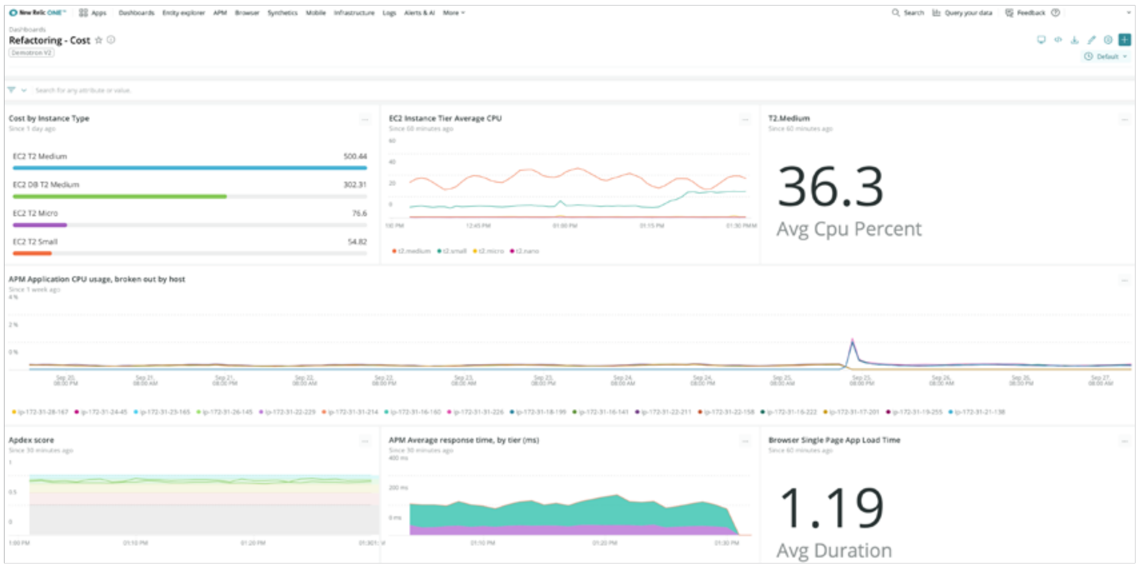


Figure 4. Track business, application, operational, and other KPIs.

Which Type of Refactoring Should You Use?

For those familiar with code refactoring, refactoring on AWS will feel familiar. However, it’s not just about refactoring code. It’s also about taking advantage of cloud technology to fundamentally:

- Reduce the overall code footprint
- Decrease the surface area that you need to secure
- Automate more of the management of your systems and reduce errors
- Achieve cost optimization
- Increase efficiency and performance

In the AWS environment, there are three different types of refactoring you can leverage for your application: code, deployment, and technology refactoring.

Best Practices for Measuring Your Code Pipeline

To learn how to use instrumentation to push changes through production more quickly, watch this webinar presented by New Relic DevOps solution experts.

Traditional Refactoring: Improving Code

Going back to its traditional meaning, the first type of refactoring is all about improving an application’s code. The idea is to identify portions of the application where it makes the most sense to re-architect the code for quality, maintainability, performance, and predictability. It’s an opportunity to fix existing issues and create less complex and more streamlined code.

The overarching questions are, “How do you know which portions of the application should be refactored? What will deliver the biggest benefit for the effort involved?”

The answer is in the data you’ve been tracking about the application. By integrating data from your code pipeline into New Relic One, you can see how changes to the codebase impact operations. Using the data gathered from the “before” stage of informed refactoring gives you insight into problem areas and opportunities for refactoring. By reviewing the baselines you’ve collected, you can start to develop a list of application areas to improve.

Next, review the code itself. For example, a more traditional e-commerce application might have areas in the code that would be good candidates for refactoring based on the metrics you’re tracking. However, you will still need to consider the effort in context with the running state of the application and end-user experience to determine whether an area will have a material impact and the corresponding positive outcomes for which you’re aiming.

Refactoring for a Different Deployment Model

After you’ve decided on the parts of the codebase to work on, the next step is to consider a newer deployment model. Choose the deployment model that makes the most sense for your organization based on your IT philosophy and future direction.

When you refactor the code, you could decide to deploy it back into a self-managed server. Alternatively, you could use AWS services to handle more of the low-value tasks for you instead. For example, you could deploy using:

- AWS Elastic Beanstalk for ease of use, deployment, and scaling of applications
- AWS ECS/EKS for Kubernetes containerization
- AWS Fargate for a fully managed cluster environment
- AWS Lambda to run code without provisioning or managing servers (serverless model)

Decisions about deployment models will impact the amount of maintenance required in the future. As you move more of your workloads to the cloud, it makes sense to adopt more advanced services and technologies such as serverless that reduce the amount of undifferentiated work needed to support your applications.

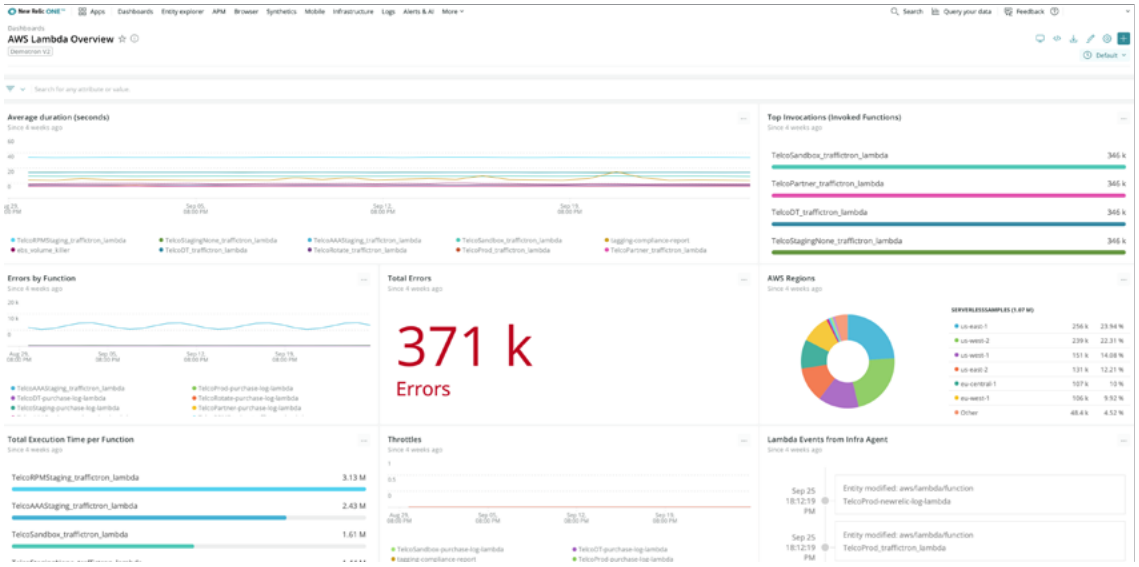

AWS Lambda Overview
Average duration (seconds)
Top Invocations (Invoked Functions)
TelcoSandbox_trafficctron_lambda
TelcoPartner_trafficctron_lambda
TelcoDT_trafficctron_lambda
TelcoStagingNone_trafficctron_lambda
Errors by Function
Total Errors
371 k
Errors
AWS Regions
us-east-1
us-west-2
us-west-1
us-east-2
eu-central-1
eu-west-1
Other
Total Execution Time per Function
TelcoRPMStaging_trafficctron_lambda
TelcoAAAStaging_trafficctron_lambda
TelcoSandbox_trafficctron_lambda
Throttles
Lambda Events from Infra Agent

Figure 5. Monitor AWS Lambda KPIs with New Relic One.

A Look at Technology Refactoring

Beyond deciding what to change and improve in your application’s code base, it’s important to also consider how new technologies can help reduce the number of to-do’s on your IT list. While optimizing your infrastructure is best done as part of a rehosting or replatforming effort (see our white papers on rehosting and replatforming to learn more), when it comes to your data store, there are many benefits of refactoring the application to take advantage of newer technology.

In the past few years, the belief that the relational database management system (RDBMS) is the only way to hold data has become outdated. Today, there are diverse data stores that support how an application uses data, including these from AWS:

- Amazon DynamoDB, a key-value document database
- Amazon Elasticsearch Service, a fully managed service for quickly searching your data
- Amazon Quantum Ledger Database (QLDB), which provides centrally managed, transparent, and cryptographically verifiable transaction logs

Five Design Principles for Cost Optimization in the Cloud

1. Adopt a consumption model (serverless).
2. Measure overall efficiency.
3. Stop spending money on data center operations.
4. Analyze and attribute expenditure.
5. Use managed services to reduce cost of ownership.

As part of your informed refactoring approach, review the data you’ve been collecting about your application to help you understand where you might take advantage of these AWS technologies. For example, if your application needs massive scale and a distributed approach, it may be beneficial to refactor to a NoSQL database service such as Amazon DynamoDB. Similarly, if several of your queries use the “like” clause, Amazon Elasticsearch might be a good option for refactoring.

For any technology refactoring you’re planning to do, it’s critical to ensure that the benefits to the business, IT, and end users will outweigh the cost and time you’ll invest in making the change.

Refactoring Within the Context of the AWS Well-Architected Framework

The AWS Well-Architected Framework helps you understand the pros and cons of decisions you make about your applications on AWS. The Framework includes architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. Together with New Relic One, the most powerful cloud-based observability platform, the Framework provides a way to consistently measure your applications against best practices and identify areas for improvement.

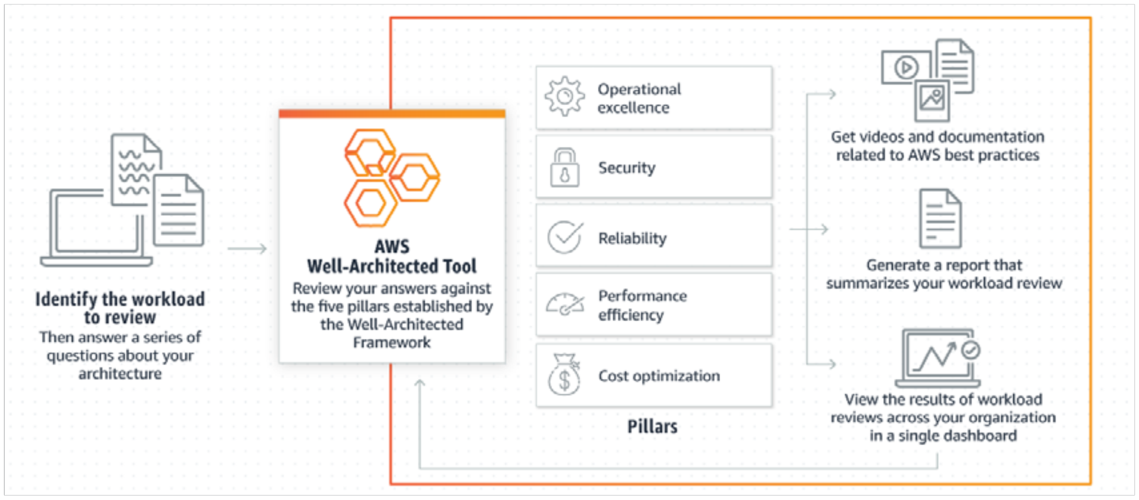

Identify the workload to review
Then answer a series of questions about your architecture
AWS Well-Architected Tool
Review your answers against the five pillars established by the Well-Architected Framework
Operational excellence
Security
Reliability
Performance efficiency
Cost optimization
Pillars
Get videos and documentation related to AWS best practices
Generate a report that summarizes your workload review
View the results of workload reviews across your organization in a single dashboard

Figure 6. The AWS Well-Architected Framework.

1. Cost optimization

Cost optimization is not simply about the amount that is spent, but the impact of the application spend. By incorporating business data via custom events in New Relic One, you can track revenue coming from customers on your website. By breaking this down by geography in New Relic One, you can gain insight into where your investment in performance improvements would have the biggest impact for your customers.

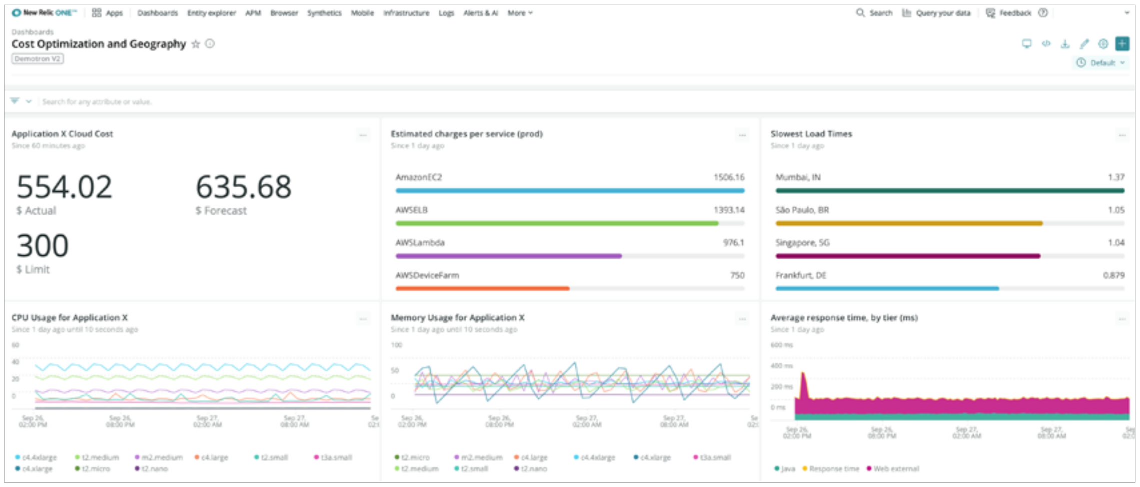


Figure 7. Track your AWS spend side by side with infrastructure, application, and geography KPIs.

For example, performance in Brazil might not be as good as you want it to be, but it might not make sense to invest in that market because it isn’t one of your top targets. Japan might have very few end users currently, but your organization wants to invest heavily to grow business in that geography. Therefore, you need to prioritize the experience for users in Japan. This combination of performance data and company objectives should influence how you determine appropriate investments.

Another way to see if you’re optimizing costs is to review your application’s Apdex score in New Relic One. Does it make sense based on your spend for that application? Or are you over-provisioning (and over-paying) to maintain happy customers at your current Apdex score?

2. Performance efficiency

This pillar within the Framework is less about application performance and more about resource usage and how quickly and easily you can change things. How easy is it for your developers to try out new technology based on how you have deployed it? Can you refactor quickly?

For refactoring to work, you need automation and infrastructure-as-code in place to help you keep up with today’s increasingly difficult demands. You can assess how well your organization is doing in these areas by monitoring and tracking:

- Deployment frequency
- Deployment success
- Time to provision an environment

Six Design Principles for Operational Excellence in the Cloud

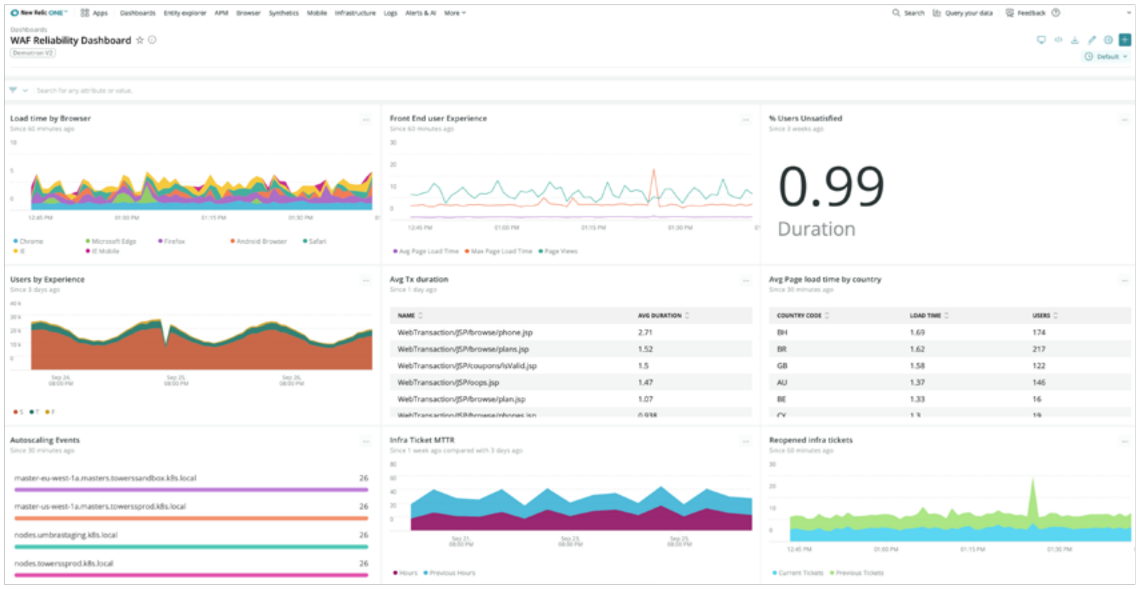
1. Perform operations as code.
2. Annotate documentation.
3. Make frequent, small, reversible changes.
4. Refine operations procedures frequently.
5. Anticipate failure.
6. Learn from all operational failures.

Understanding how these metrics change after a refactoring will be a leading indicator of having made the right refactoring choices.

3. Reliability

Reliability goes beyond whether your application is up or down. It’s equally important to track and understand how well your application is scaling during peak events. What happens when your traffic spikes? How good is your testing? Are issues found in development, test, or production? What is your mean time to resolution (MTTR)?

With refactoring, the goal is always to achieve greater stability after the project. Automation plays a large part in improving stability because it reduces the risk of human error. That’s why after a refactoring effort, your organization should see not only fewer issues, but should be identifying those issues earlier in the application life cycle.



NAME	AVG DURATION
WebTransaction/JSP/browse/phone.jsp	2.71
WebTransaction/JSP/browse/plans.jsp	1.52
WebTransaction/JSP/coupons/isValid.jsp	1.5
WebTransaction/JSP/oops.jsp	1.47
WebTransaction/JSP/browse/plan.jsp	1.07
WebTransaction/JSP/browse/phones.jsp	0.938

COUNTRY CODE	LOAD TIME	USERS
BH	1.69	174
BR	1.62	217
GB	1.58	122
AU	1.37	146
BE	1.33	16
CY	1.3	19

master-eu-west-1a.masters.towerssandbox.k8s.local	26
master-us-west-1a.masters.towerssprod.k8s.local	26
nodes.umbrastaging.k8s.local	26
nodes.towerssprod.k8s.local	26

Figure 8. A dashboard showing the Well-Architected Framework, application, and end-user KPIs.

4. Operational excellence

This pillar is about improving operational processes and procedures, team performance, and collaboration. It’s also about fostering a culture where experimentation is embraced. To track improvement, this requires monitoring not just application performance, but the application’s codebase and the use of infrastructure-as-code to automate the operation of the application.

Focus on how well your team is using infrastructure-as-code to improve and automate processes. You can do this by monitoring indicators in your cloud environment, such as the number of AWS CloudFormation deployments of specific templates and any errors associated with those deployments. Also, keep an eye on how often your repository of Terraform or CloudFormation code changes. Where are issues being found—in development, test, or production?

5. Security

This principle is about improving your security posture. One goal of refactoring is to reduce the amount of code to maintain, which reduces complexity, testing, and the likelihood of security flaws. Another goal of refactoring for AWS is that your organization has fewer technologies that you must manage yourself. This reduces the amount of infrastructure for which your organization is responsible for vulnerability and patch management. Less code to maintain and less infrastructure to patch improves your security posture by reducing the attack surface that your organization needs to protect.

Conclusion

As an introduction to the topic of refactoring applications for the AWS environment, we’ve covered the basics of why and how a refactoring approach based on observability can help you achieve the highest outcomes and demonstrate the impact of your efforts on the application, your customers, and your business. We’ve discussed how you can use New Relic One to determine what and when to refactor to continually modernize your applications for the greatest benefit, using data about your:

- Infrastructure
- Application running state
- End-user experience
- Business KPIs
- DevOps processes
- Codebase changes and issues

One final word about refactoring: It’s more than a single, one-time project. View it as the ongoing process of continuously making your applications better in order to improve your performance, customer experience, and business outcomes.

To learn how New Relic One can help you in refactoring or any phase of application and infrastructure modernization, visit newrelic.com/aws.

New Relic.