aws

WHITE PAPER

Optimizing Your Cloud Migration

Paths to consider when moving your Oracle workloads

In collaboration with

ONICA by rackspace technology

Business imperatives to migrate data to the cloud

To boost innovation, respond quickly to changing demands, and drive business transformation, organizations are migrating and modernizing their infrastructures and applications on Amazon Web Services (AWS). Through programmable infrastructure and managed services, AWS enables modern operational practices that lead to measurable results. On average, migrating to AWS delivers:

20% average infrastructure cost savings[1]

66% increase in administrator productivity[1]

43% faster time to market for new features[1]

29% increase in staff focus on innovation[1]

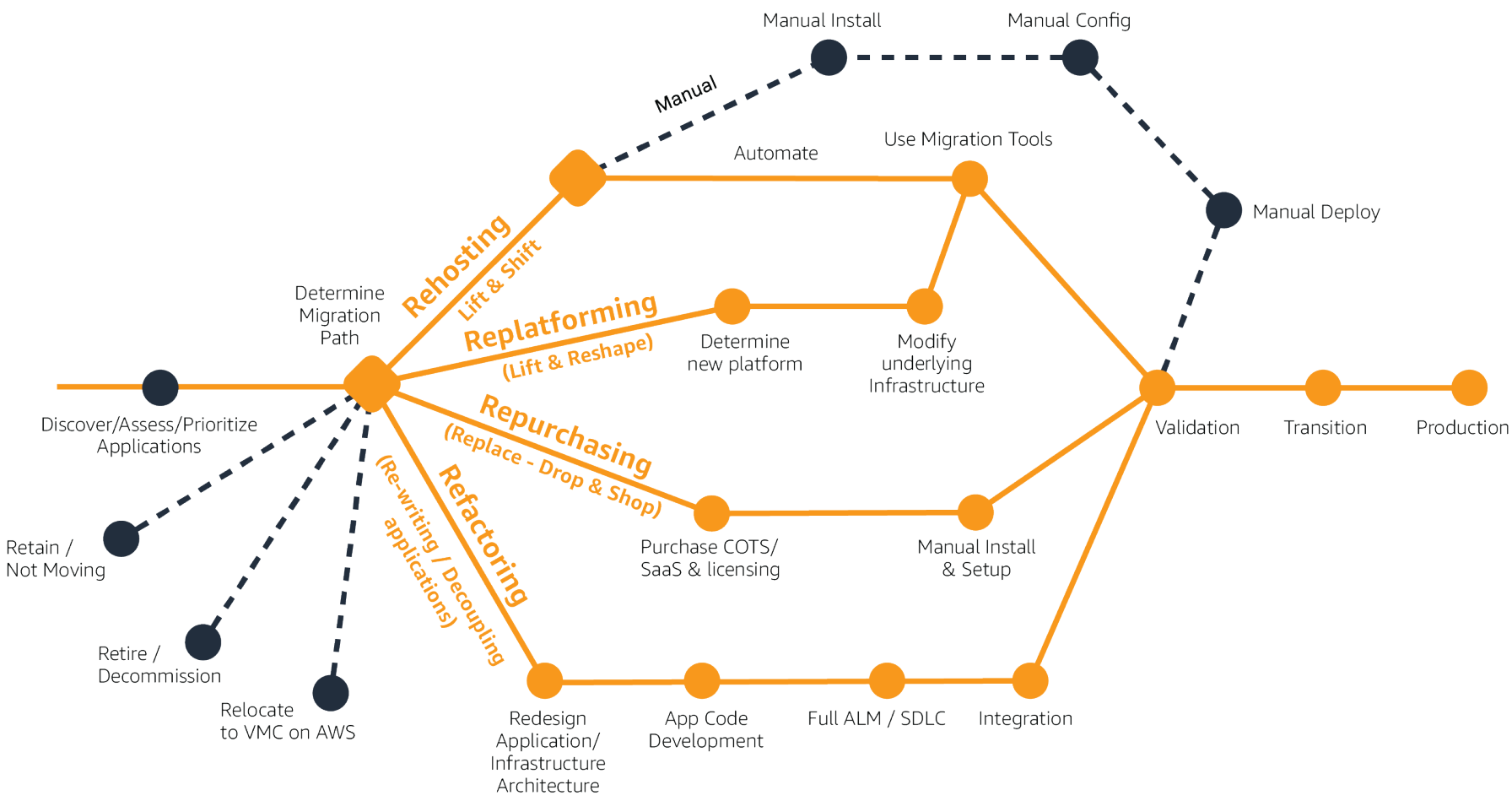
45% fewer security-related incidents[1]

Clearly, migrating to AWS is good for business. But successful migrations take planning and expertise, as well as an understanding of the challenges you're likely to face as part of the process. By understanding those challenges, the pitfalls that can result when they aren't fully addressed, and the possible solutions to smooth your way forward, you've taken the first step on your cloud migration journey.

1 "The Business Value of Migration on Amazon Web Services," The Hackett Group, 2022

aws

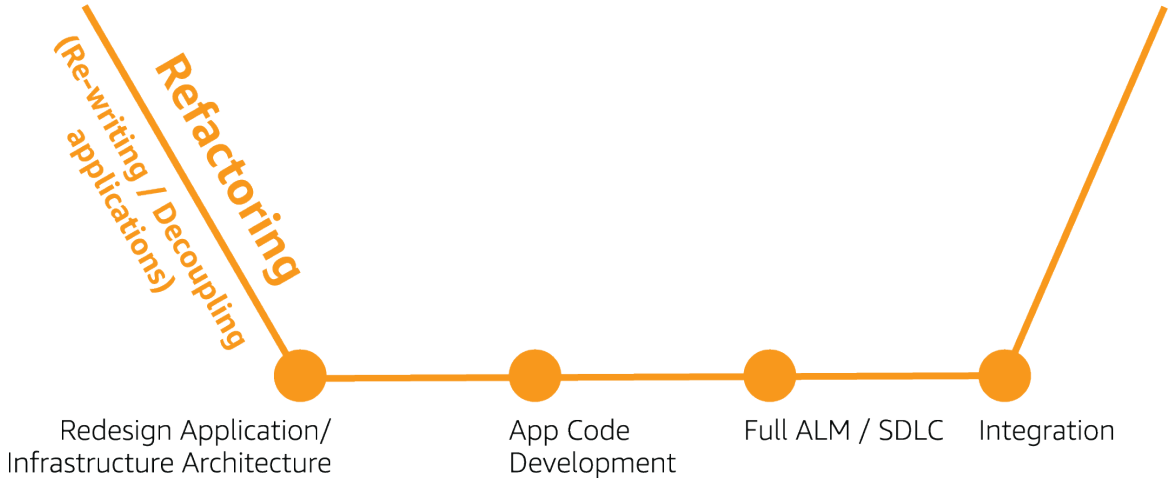
The key migration paths



Refactor for cloud-native design

Refactoring for cloud-native design is the path that requires the most forethought, planning, engineering effort, and time overall to implement, but benefits from being the most stable solution wherever possible. Refactoring existing applications as they are currently used and written allows for the option to fully automate the build/test/release cycle processes (leveraging more modern CI/CD pipelines), and ensures that old workloads get updated to modern processes and architectures. The opportunity to take an old, problematic application and re-write it to be cloud-native may include serverless options, the opportunity to incorporate native backup and restore solutions, disaster recovery or highly distributed design and architecture, and anything else that would allow you to benefit from the services provided by the public cloud. Often this includes moving away from existing data structures toward a more modern database engine, providing an opportunity to escape expensive licensing by choosing open-source standards, etc.

Refactoring



Planning around this requires extensive research to identify and resolve automated processes around every aspect of the workflow. Concepts common to this model would require DNS automation, service discovery, blue-green or canary deployment, centralized logging, version control workflows, artifact creation and storage, etc. —the list goes on, and there are a lot of things to think about. A high level of maturity is strongly recommended for making these decisions, whether that's internally or through the use of a trusted partner. This is the essence of "Migration as Code" and is the primary way Onica by Rackspace Technology operates its migrations.

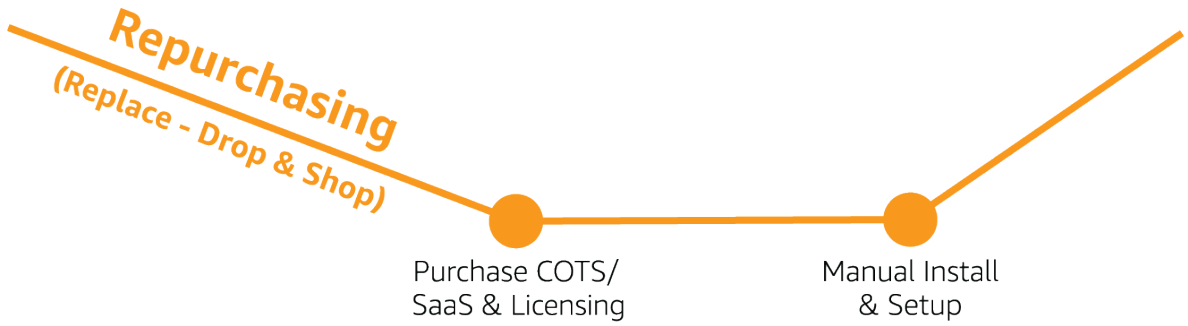
This is not necessarily always an option, however. The major factors that play into this option are the obvious ones: time and money. An organization may not have the engineering resources or maturity to implement a custom, cloud-native solution to an off-the-shelf application currently in use today. Even in-house applications may not be worth refactoring within the given time, or other business priorities may make it an impossibility. Without available engineering resources, the company is left in a position to hire new talent with those skills, train their existing workforce and expect slower delivery of the solution, or simply choose an alternative path.

Automate with cloud-native architecture

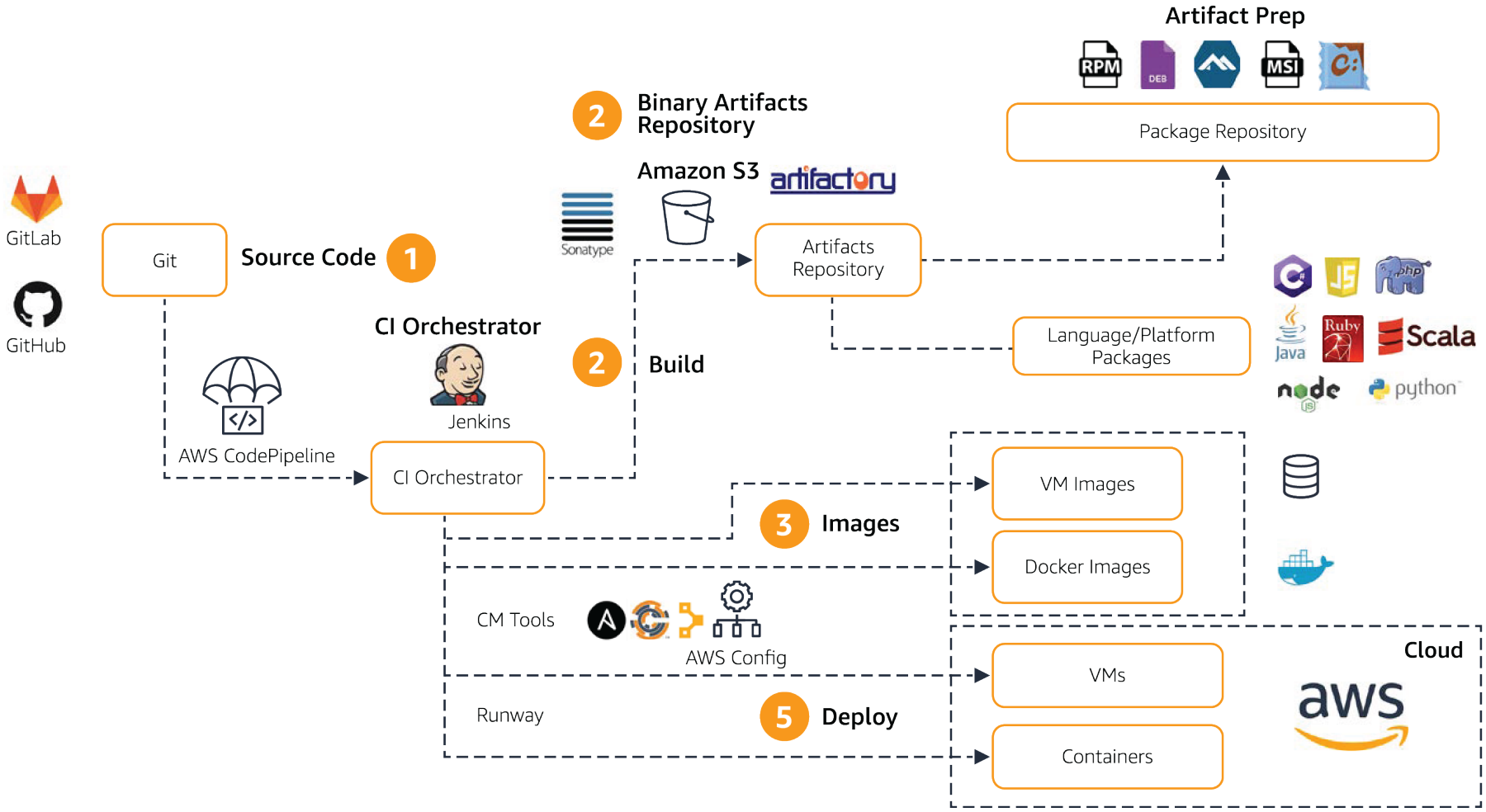
Ideally this solution is also paired with CI/CD deployment processes to solidify process and automate concepts throughout the organization, allowing for simpler onboarding processes for other applications that can follow this model. Most often this process will include several phases of pipelines, will perhaps make use of baseline machine images for ready-use in infrastructure templates, and will leverage a deployment mechanism for applications onto these machines post-creation.

Straying a little further away from a full refactor is the idea of taking the same applications and mindsets, but migrating them with more cloud-native architecture. This means using automation from start to finish, codifying all infrastructure, and designing with the benefits of the public cloud in mind for availability and disaster recovery. An example of this would be to take a workload in your data center today and write functional code to deploy its infrastructure and application installation into the cloud in such a way that it's highly available (self-healing with autoscaling groups and effective health-checks).

Repurchasing



An overview of pipeline-driven migration



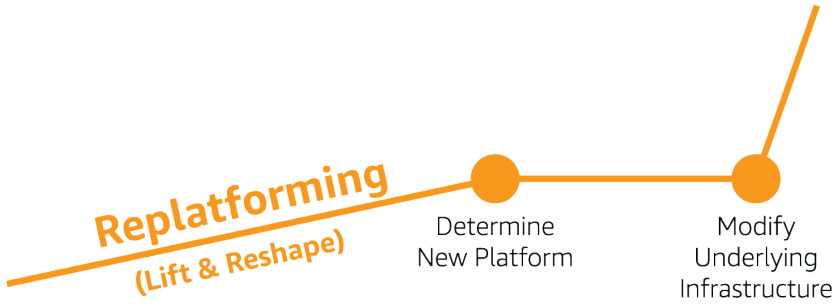
Many, if not all, of the external processes that would need to be identified for this strategy are the same as a full refactor, as things like DNS and service discovery may prove to be necessary components for the levels of automation that are intended. Deployment processes need to be identified, artifacts still need to be built, and code needs to be written for all of these things to talk to each other. The Migration as Code approach is still key here, and Onica uses this approach when other business priorities make full refactoring a challenge.

This is a very common path to take as it requires less engineering effort than a full refactor and still allows for many of the benefits of the cloud. Blockers to this process may come in the form of licensing limitations for the workload, the application behavior or installation, or limited time and resources. For example, it may be relatively simple to automate the installation and configuration of a web application running in Apache on Ubuntu simply because of the nature of the operating system, its scriptability, and the configuration processes required. However, an off-the-shelf application running on Windows may be packaged in such a way that the installation cannot be fully automated despite all the neat tricks in your toolbelt or may require that the vendor log into the server after installation in order to manually license the application (yes, this happens), or any number of factors that make it impossible to automate the process end-to-end. Workarounds can likely be identified for some of these issues, but it all depends on the amount of time and energy the organization wants to sink into this process before cutting losses and choosing another path.

Partial automation

Partial automation is a further step back. In situations where fully automated processes simply can't be achieved, the concepts of "infrastructure as code" and partial automation step in to bridge the gap.

Replatforming

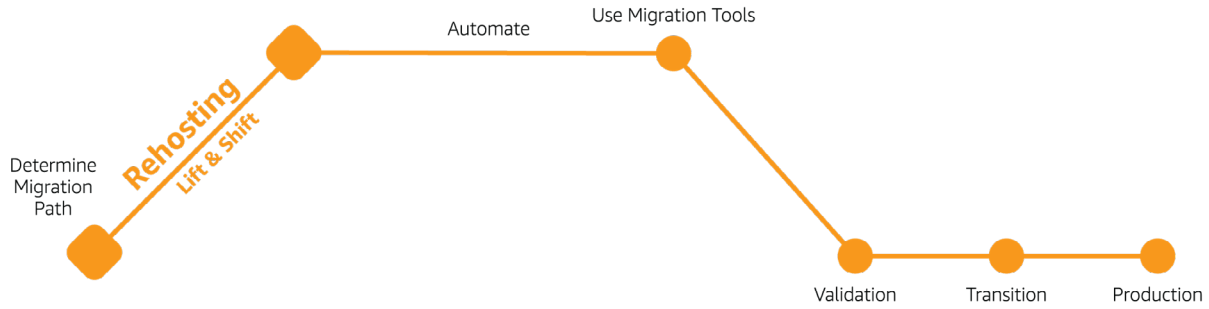


Infrastructure code can be written to deploy resources such as servers, load balancers, etc., but in this path manual intervention is required to install and configure the workloads, databases, and glue in between those components. An example of this might be a common workload "farm" at an enterprise: AWS CloudFormation or Terraform may be written to deploy a collection of servers in the server farm, database servers running a given version of the database engine, load balancers pre-configured with a few listener rules, and security groups to ensure connectivity between resources. From here, a systems administrator or application owner would log into the servers, run installation scripts if possible, and manually configure connectivity between nodes by including static IP addresses or hostnames, etc.

This is still a beneficial path in the event that the migration path is very large. Oftentimes this automation, while not end-to-end, can still be leveraged for its generality. The same infrastructure code to launch an eight-node Windows farm with a load balancer can likely be used to deploy a six-node Linux cluster intended for a Cassandra installation, with only a few tweaks to the number of nodes, the machine image used, and a yes/no option for the load balancer. Even incorporating this level of automation can help accelerate additional effort in the future, and provides a starting point for later ventures into the more complex automated paths. The downside of this path is that things must be done manually. This means that instead of treating your services like nameless, faceless resources that do their job on their own, you're stuck in the older data-center-centric model of naming your servers, taking care of them, and troubleshooting them when they fail. While this may sometimes be a necessary approach, the Onica team still tries to avoid it when possible. Our belief is that using a model that acts as a data center prevents you from getting the exciting futuristic benefits of the cloud, impacting the overall value to you.

Lift and shift

Rehosting



This migration path uses some sort of third-party tool, typically to create a duplicate machine as-is currently running on-prem or in colocation, sends it to the AWS cloud as a machine image, and then launches the server in AWS. It is essentially an identical, block-for-block clone of the original machine. Often the tool used for this process provides simple interfaces for managing this process, and can synchronize data changes as they happen on the live machine up to the virtualized image. This is a particularly ideal choice with the “black box” situation, a server that has been around since the dawn of the company running exceptionally important business logic, except that nobody knows how it works or what functions it performs. As long as the baseline requirements are met for the migration tool, you can ship it up to AWS and turn it on, and it’ll be the same box as on-premise. When it comes to speed and minimal level of expertise required, this is generally the easiest way to go, with some significant caveats. The tools that offer this migration path are very limited. They can achieve their task (usually) but can’t handle variation on that process. They also cannot be automated, as the target demographic tends to prefer Graphical User Interfaces over scripting languages or command-line options, so commercially this is where the vendors spend their development energy. These tools can often be very expensive, and require licenses to be purchased per server slated for migration.

Data synchronization may not be able to keep up with highly active changes, and we've seen multiple cases where a database was started with this process and could never keep up with shipping the delta up to the cloud, eventually falling farther and farther behind until a new migration strategy had to be implemented. Finally, sometimes things just don't work, the image doesn't boot for mysterious reasons, and nobody (even the vendor) seems to know why, and you just have to try again or find another solution.

Most commonly, the issue with this migration path tends to be the inability to test effectively in Windows environments, tied to the use of Active Directory. When a Windows instance joins Active Directory, it defines several factors about itself that just aren't very "cloudy": hostnames often need to stay static for DNS and application-config purposes, the operating system sets several unique identifiers in the Windows registry that specifically define what this machine is and what it does, etc.

All of these things mean that if a second machine were to come up at the same time and connect to Active Directory, there would be duplicate machines, and things will just not work right. Your SysAdmins will have a terrible time troubleshooting the issues and you may end up with production outages. This means that testing the generated machine image needs to happen in a bubble without connectivity to Active Directory, which usually means your application doesn't work and testing is useless. All actual testing really needs to happen during cutover, while the production instance is down and disconnected, leading to even longer
outage windows.

Data migration

Data is a special factor in all these migration paths, inevitably centered around what the expected downtime during cutover can be. With very large Recovery Time Objectives, a database can be taken offline, backed up, the backup sent to a new database server in AWS, and traffic shifted to this new server. With very low RTO, the ideal scenario is to attempt to expand the database cluster by adding a replica in AWS, ensuring streaming replication in near-real-time, and then cutting to the replica using the database's own failover mechanisms. These capabilities are limited to the database engine in use, and potentially the licensing level of that database, so always be sure to perform discovery ahead of time for any of these factors that may come into play when making a decision to migrate a database.

For a migration from Oracle database to AWS there are three common strategies: rehost, re-platform or re-architect. Rehosting is migrating your Oracle Database without changing the operating system, software or configuration and utilizes Amazon EC2. Re-platforming is moving to a DBaaS offering such as Amazon RDS for Oracle. And re-architecting (refactor) is best used when you want to re-architect your database and application to utilize open-source and cloud-native database services such as Amazon Aurora PostgreSQL-Compatible, Amazon Aurora MySQL-Compatible or MariaDB.

Tools which convert data from one database engine to another do exist, allowing for migration out of one engine and into a comparable platform (like AWS DMS and other tools specific to Oracle migrations). However, these tools tend to manipulate the data, and may have technical limitations that make the data unusable in the new engine, so this must be tested ahead of time. Another issue with this method is that it requires testing of client applications to ensure proper connectivity to the new database engine.

.

Database modernization

Early cloud adoption approaches focused on migrating applications and infrastructure with minimal changes to the code or architecture. With a data modernization strategy, organizations can move data from legacy databases to modern database modernization and unlock cost savings and the improved performance and scalability gained with moving to AWS. A database modernization allows you to save on licensing costs from legacy databases and benefit from serverless technologies. Other benefits of data modernization include improved reliability, increased scale, develop at cloud scale, and increase agility so that you can accelerate innovation.

Modernization strategies include:

- Assessment with recommended custom migration solution aligned to your business goals and timeline.
- “Lift and Shift” from on-prem or private cloud environments to AWS.
- “Move to Managed” by migrating Oracle, SQL Server, MySQL, and PostgreSQL databases to fully managed Amazon RDS.
- “Break Free” from commercial and proprietary databases with database transformations to Amazon Aurora.

Retail case study

Challenge

Reduce Oracle licensing costs through a migration while also reducing planned downtime during the migration.

Solution

With support from AWS partner Onica, by Rackspace Technology, the customer determined the databases that needed to be re-platformed to Oracle RDS to maximize the benefits of moving to AWS and reduce costs with Oracle licensing.

Results

Onica reduced licensing costs from several millions of dollars with the reduction of Oracle licensing, down to the hundreds of thousands on AWS including Infrastructure.

In addition, Onica met the customer's current performance needs and opened opportunities to expand load without a significant licensing cost increase.

Conclusion

The overall concept of these points is simply that there is no single tool for a migration, and care must be taken to identify the proper path for each workload to align it with business objectives. At Onica, we always recommend automating as much as possible, because reusability is key when it comes to any technical process.

About Onica by Rackspace Technology

Onica by Rackspace Technology is a dedicated Amazon Web Services (AWS) business unit at Rackspace Technology. As a Premier Consulting Partner, Onica helps customers drive innovation, agility, cost savings and operational efficiency on the AWS Cloud. Onica is focused on bringing the most cutting-edge AWS capabilities to every customer through deep expertise in AWS strategy, cloud native development, containers, application modernization, AI and machine learning, and IoT. With in-depth advisory services built on 15 competencies and experience with 1,000+ customer launches, Onica by Rackspace Technology is here to help you accelerate cloud native transformation on AWS.

onica@rackspace.com

In collaboration with

ONICA
by rackspace technology.