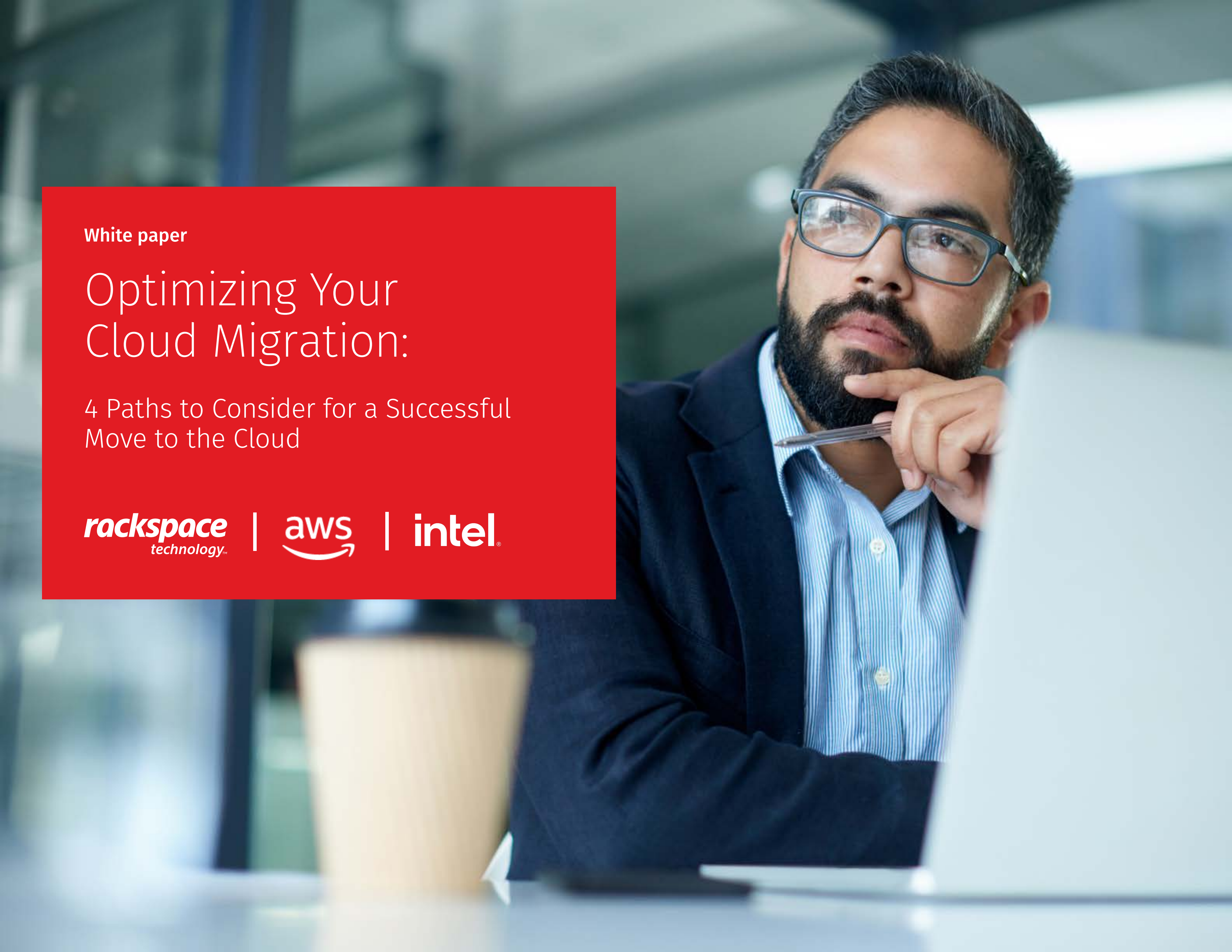


White paper

Optimizing Your Cloud Migration:

4 Paths to Consider for a Successful Move to the Cloud

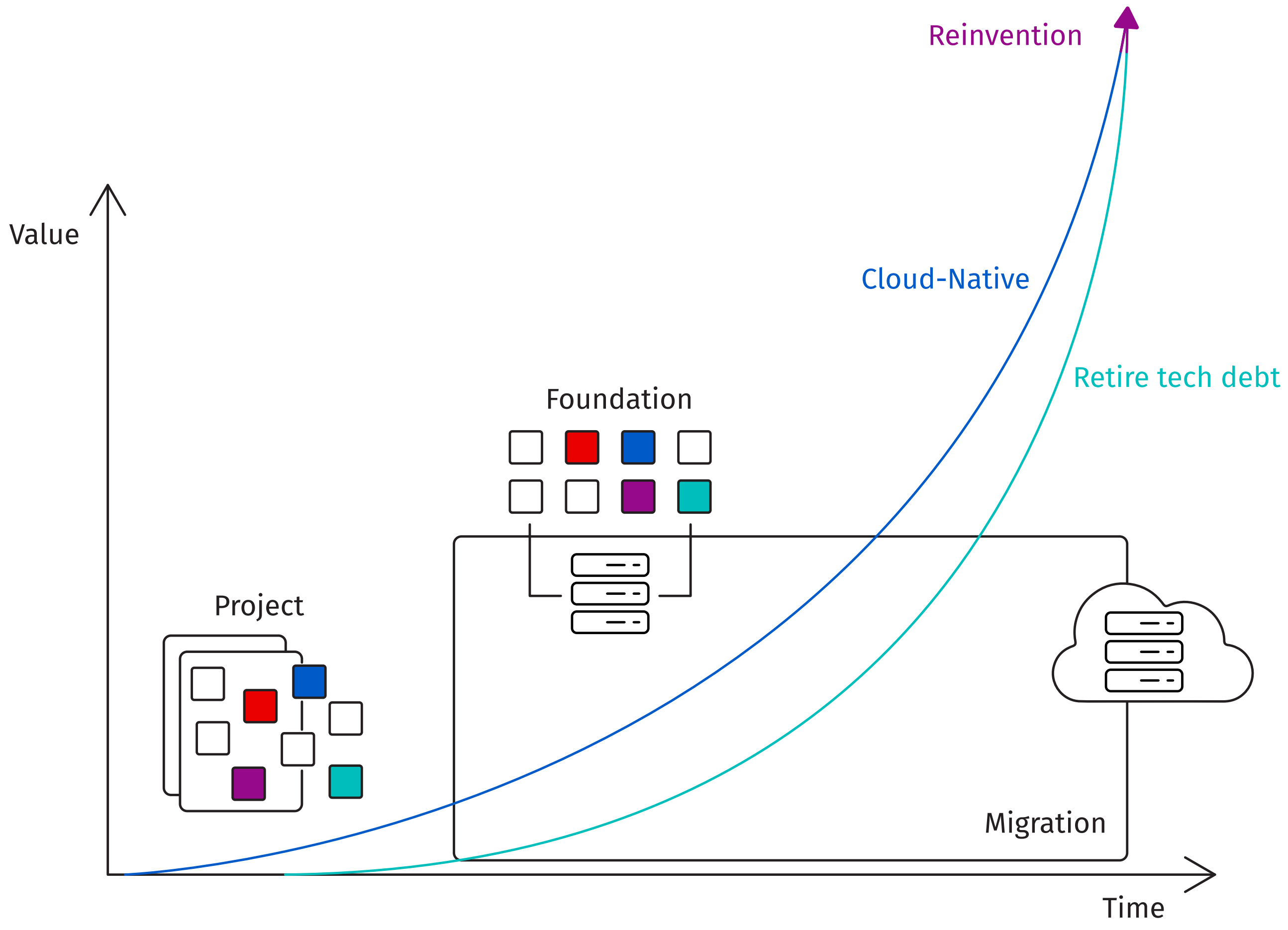
Optimizing Your Cloud Migration

A company can take many paths when moving to the public cloud. Deciding how to migrate to AWS should be a process that focuses on each individual workload, rather than trying to find a single solution to be used across the board. Key to these decisions are factors around:

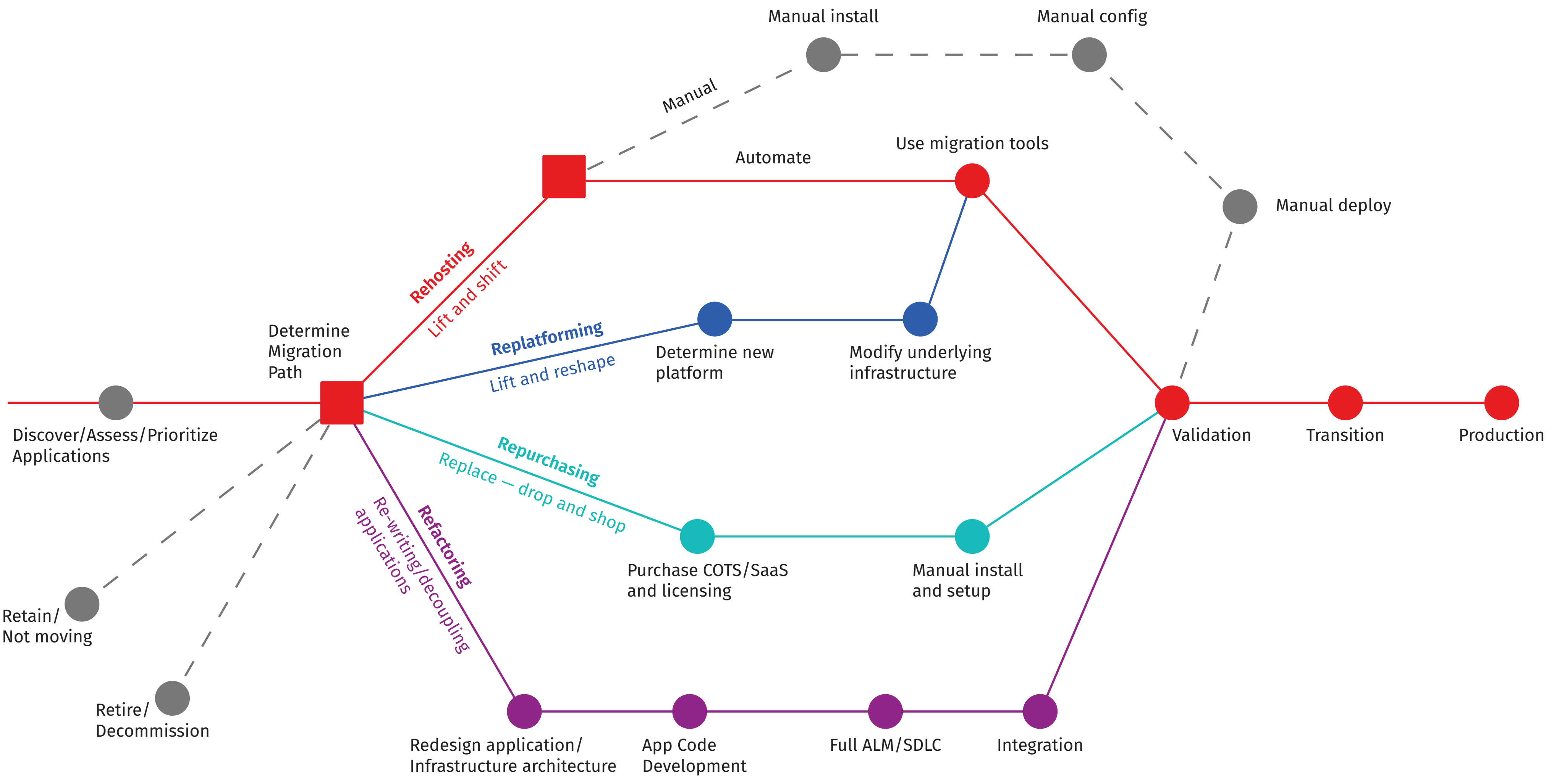
- Level of engineering effort
- Time constraints
- Licensing restrictions
- In-house vs. Commercial-Off-The-Shelf (COTS) software
- Resource requirements

With each of these limitations in mind, there are a few paths that should be assessed for viability as you decide to make your migration plans. Rackspace Technology always recommends assessing these paths in this order, as they are organized from most-to-least ideal.

If a path is not possible, scrap it and move on to the next. It's important to remember that these paths are not interchangeable. Rather you should carefully consider everything before eliminating a path from your process.



The Four Migration Paths



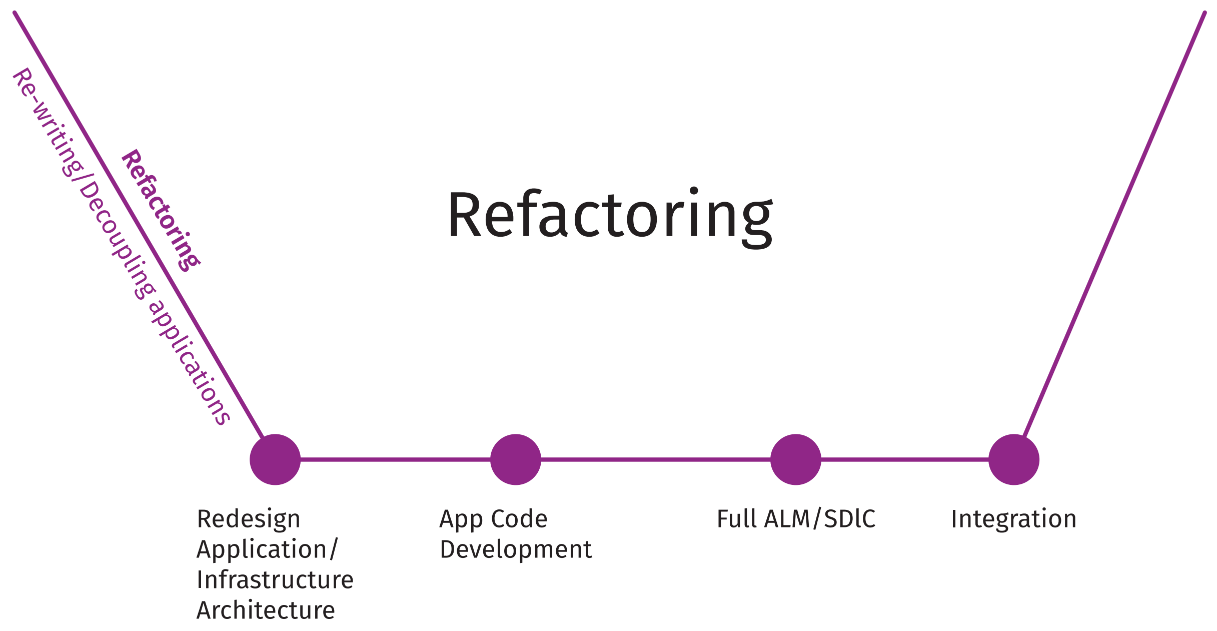
Refactor for Cloud Native Design

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 overall.

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 even 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.

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 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.

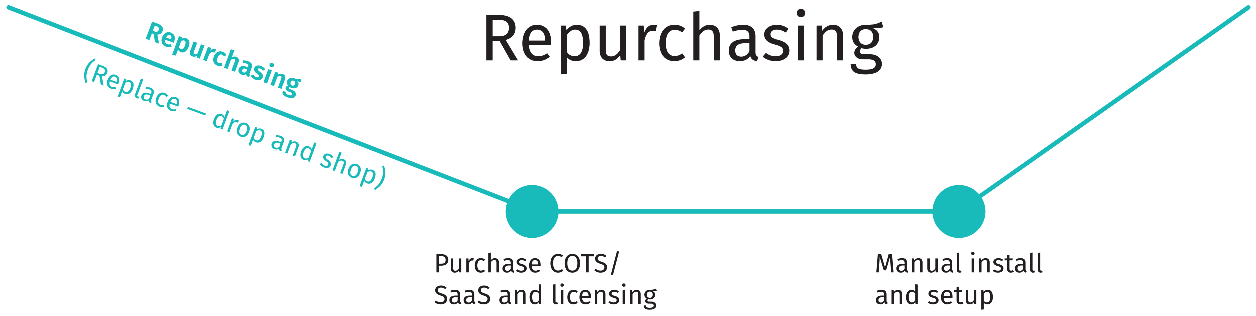

Refactoring
Re-writing/Decoupling applications
Refactoring
Redesign Application/ Infrastructure Architecture
App Code Development
Full ALM/SDlC
Integration

Automate with Cloud-Native Architecture

Ideally, this solution is also paired with CI/CD deployment processes to solidify process and automate concepts inherently 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).

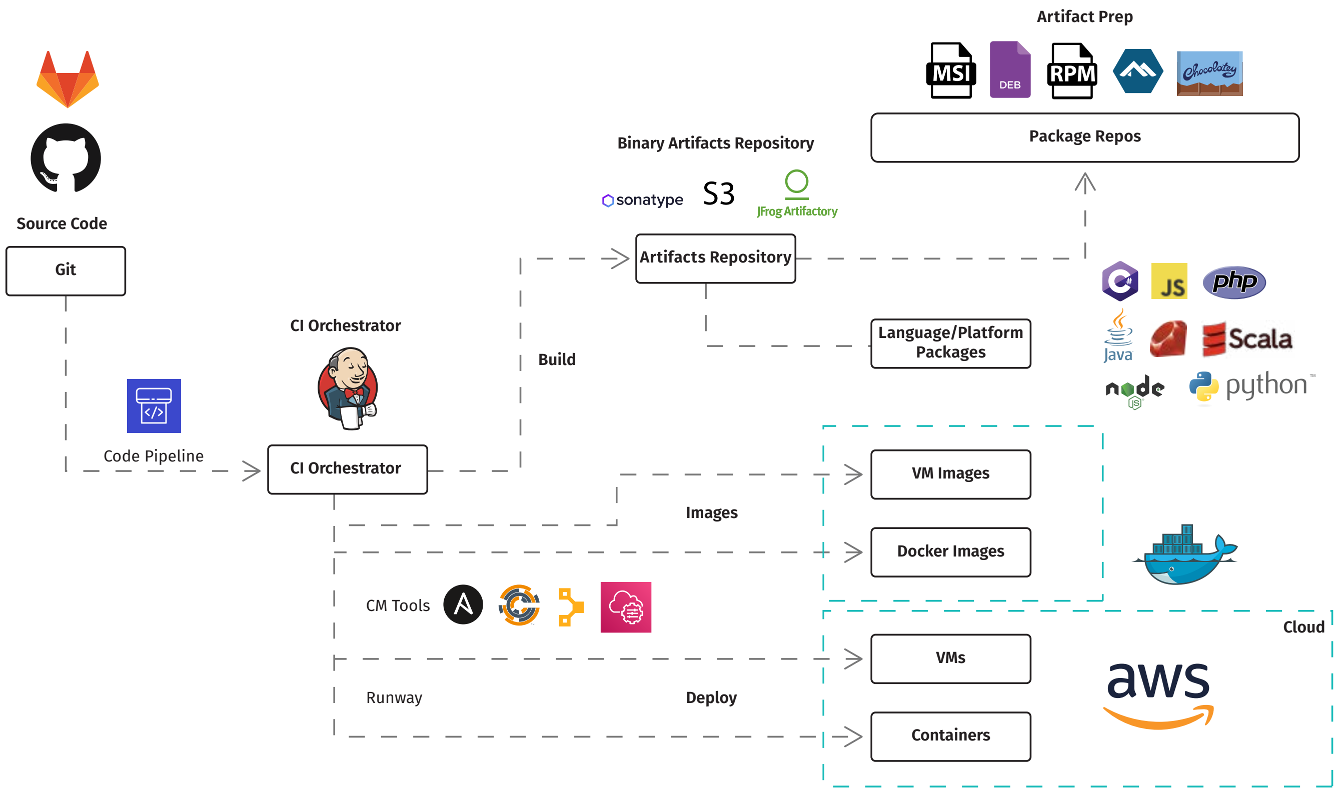
Repurchasing



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 Rackspace Technology 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. Obstacles to this process may come in the form of licensing limitations for the workload, the application behavior or installation, or again the obvious factors of 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

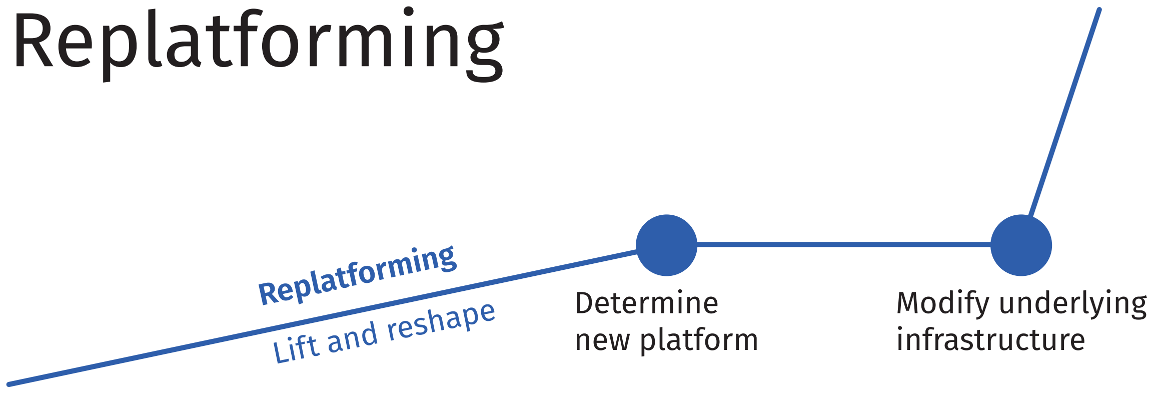
Partial automation is a further step back. In situations where fully automated processes simply can't be achieved, the concepts of "infrastructure as code" and partial automation step in to bridge the gap.

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, 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. Often times 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 therefore help accelerate additional effort in the future, and provides a starting point for later ventures into the more complex automated paths.

The downsides of this path are obviously the fact that things are being done manually. This inherently 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, Rackspace Technology 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. Instead, we've done more of just moving our datacenter to somewhere else, but we have a lot of exciting scripts that we've written, and can hopefully use what we've learned writing them to move forward to more automated processes!

Replatforming



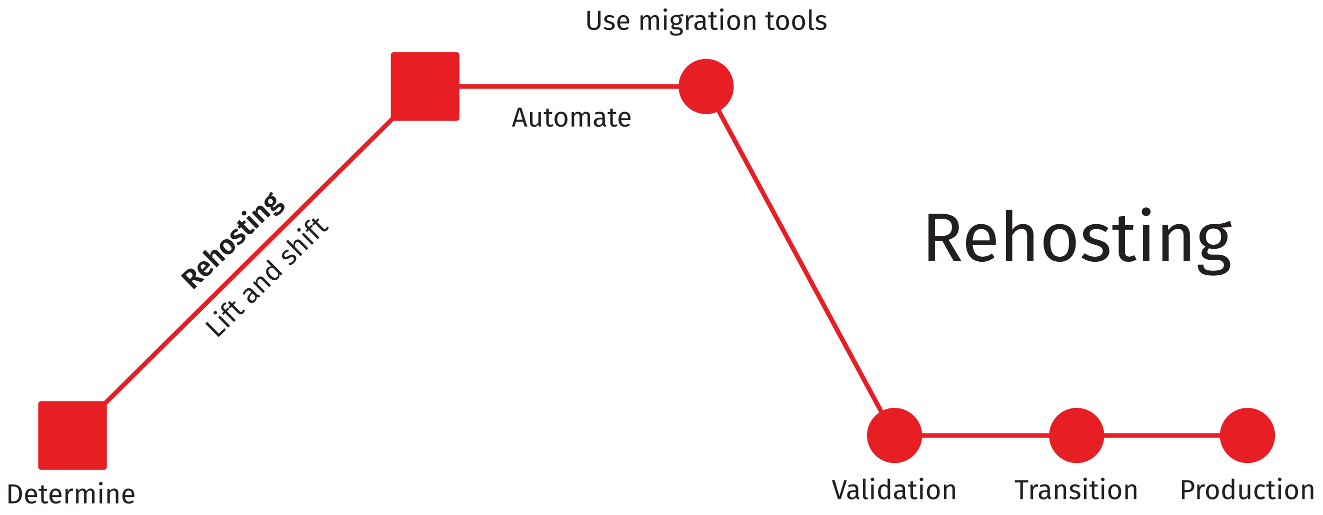
Lift and Shift

This migration path uses some sort of third party tool typically to create a duplicate machine as is currently running on-premises 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 typically 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 nobody knows how it works or what it does exactly. 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-premises. 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. At the end of the day, Rackspace Technology recommends avoiding a Lift and Shift migration at all costs because not only will there be inevitable problems in your migration, but you won’t be able to utilize the full potential of the cloud.

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, Active Directory would become angry, and things will just not work right. Your SysAdmins will have a terrible time troubleshooting the issues and you'll probably 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 even work, which means the testing is pretty useless. All actual testing really needs to happen during cutover, while the production instance is down and disconnected, leading to even longer outage windows.

Honorable Mention: 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 any of these factors that may come into play when making a decision to migrate a database.

Tools which convert data from one database engine to another do exist, allowing for migration out of one engine and into a comparable platform (MS SQL to PostgreSQL is very common). 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.

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 Rackspace Technology, we always recommend automating as much as possible, because reusability is key when it comes to any technical process.

Learn more at www.rackspace.com or call:
AU: 1800 722 577
NZ: 0800 451 613
HK: 0800 900 330
IN: 000800 100 8796
SG: 0800 120 6726
MY: 1800 812 620

About Rackspace Technology

Rackspace Technology is the multicloud solutions expert. We combine our expertise with the world's leading technologies — across applications, data and security — to deliver end-to-end solutions. We have a proven record of advising customers based on their business challenges, designing solutions that scale, building and managing those solutions, and optimizing returns into the future.

As a global, multicloud technology services pioneer, we deliver innovative capabilities of the cloud to help customers build new revenue streams, increase efficiency and create incredible experiences. Named a best place to work, year after year according to Fortune, Forbes, and Glassdoor, we attract and develop world-class talent to deliver the best expertise to our customers. Everything we do is wrapped in our obsession with our customers' success — our Fanatical Experience™ — so they can work faster, smarter and stay ahead of what's next.

Rackspace-White-Paper-4-Paths-Migration-ITT-TSK-2391 :: August 31, 2020