Replatforming your mainframe applications by using a shared IBM Db2 for z/OS database for gradual migration

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Replatforming your mainframe applications by using a shared IBM Db2 for z/OS database for gradual migration

Table of Contents

Introduction **1**
Business outcomes 2
AWS Mainframe Modernization **4**
Replatforming 4
Automated refactoring 4
Benefits of replatforming 5
The transformation process **7**
Planning 8
Application discovery 8
Data dependencies 8
Capacity benchmark 9
Wave planning 11
Building 11
Application consistency 12
Architecture 13
Running 14
Two-phase commit (2PC) 15
Runtime infrastructure 16
Testing 17
Source environment 17
Target environment 18
Analysis 18
Testing your application in AWS Mainframe Modernization 19
Cutover 20
Provision 20
Go live 20
Rollback 21
Conclude 21
Architecture 21
Best practices **23**
Network latency 23
Security 24
Application governance 24
Elasticity 25

Next steps 26
Resources 27
AWS documentation 27
Rocket Software references 27
IBM references 27
Tools 27
AWS Prescriptive Guidance patterns and guides 27
Document history 28
Glossary 29
..... 29
A 30
B 33
C 35
D 38
E 42
F 44
G 46
H 47
I 48
L 51
M 52
O 56
P 59
Q 61
R 62
S 65
T 69
U 70
V 71
W 71
Z 72

Replatforming your mainframe applications by using a shared IBM Db2 for z/OS database for gradual migration

Luis Gustavo Dantas and Andre Botura, Amazon Web Services (AWS)

May 2025 (document history)

In the constantly changing landscape of enterprise technology, mainframe modernization has become a critical requirement for organizations that need to remain competitive and agile. This transformation is not merely about replacing old systems with new ones; it's a strategic evolution that bridges the gap between the robust, reliable foundations of the past and the dynamic, innovative possibilities of the future.

The mainframe, which was once the undisputed leader of enterprise computing, is now at a turning point. Its peerless processing power and security features have kept it relevant for decades, but today's businesses require systems that can seamlessly integrate with cloud services, support mobile applications, and harness the power of artificial intelligence and big data analytics.

Modernization doesn't always require a complete migration away from mainframes. Some organizations are opting for hybrid approaches that take advantage of the strengths of both mainframe and cloud environments. This strategy enables them to maintain critical legacy applications while they gradually transition to more modern platforms. This technological transition involves more than just system updates; it requires transforming organizational culture and skills. As companies modernize, they invest in both new technologies and their workforce by bridging generational gaps and promoting ongoing learning and innovation.

This guide discusses a gradual migration strategy that balances the benefits of mainframe systems with the advantages of modern cloud technologies. This phased replatforming approach first migrates the application layer, while maintaining connectivity to your existing IBM Db2 for z/OS database to streamline the transition process and minimize disruptions to your critical business operations while you adopt new capabilities in the cloud.

This guide is designed for technical decision-makers and implementation teams that are involved in mainframe modernization initiatives. Primary audiences include enterprise and solutions architects, technical project managers, and modernization program leaders who need to understand both the strategic and the technical aspects of mainframe replatforming. The content is equally valuable for implementation teams, including mainframe application developers, AWS or cloud engineers, database administrators, and DevOps engineers, who are responsible for executing implementing the modernization.

Business outcomes

Companies have many compelling reasons to update their legacy applications. This process creates a sense of urgency across industries. When older experts retire, they leave a significant knowledge gap, which makes it crucial to modernize systems before this expertise is lost. Additionally, companies are driven by the need to reduce costs, increase agility, and respond quickly to rapidly changing market conditions.

The push for digital transformation is further intensified by emerging technologies and the demand for enhanced customer experiences. These factors, combined with the risks associated with maintaining complex systems, are prompting organizations to act swiftly in modernizing their IT infrastructure.

Mainframe modernization, in particular, presents a delicate balancing act. Companies must preserve the stability and security that mainframes are known for, while they embrace the flexibility and scalability offered by modern architectures. This process involves complex decisions about which applications to migrate, which to rewrite, and which to retain on the mainframe.

Key drivers for modernization include agility and cost reduction:

- **Agility and time to market**. Modern systems enable faster procurement processes and quicker responses to changing market demands. The adoption of DevOps and SysOps practices can significantly improve productivity and deployment speeds.
- **Cost reduction**. Modernization often leads to reduced infrastructure costs through:
  - Pay-as-you-go models, which align costs with actual usage.
  - Reduced licensing fees associated with legacy systems.
  - Improved elasticity, which provides better resource allocation.
  - Active-active, high availability setups, which enhance system resilience while optimizing resource utilization.

Based on these business drivers, COBOL application replatforming is considered a strategic approach to modernization. You can use a shared database to follow a gradual migration path that balances the need for modernization with the imperative to maintain business continuity.

This method allows you to capitalize on the benefits of modern architectures while preserving the reliability of your COBOL applications. As a result, you can achieve agility, cost efficiency, and innovation while mitigating the risks that are associated with large-scale, abrupt transitions. The shared Db2 database approach described in this guide provides a bridge between legacy systems and modern platforms, and enables a smoother, more controlled modernization process.

In this guide:

- AWS Mainframe Modernization
- The transformation process
- Best practices
- Next steps
- Resources
- Document history

AWS Mainframe Modernization

The AWS Mainframe Modernization service enables you to migrate your legacy mainframe applications to a cloud-native environment, preserve existing business logic and investments, use automated tools and managed runtime services, optimize application performance, and reduce operational costs. This service streamlines the modernization process, so you can harness the power of the cloud while maintaining the value of your core mainframe systems. AWS provides two key approaches to mainframe modernization: replatforming and automated refactoring.

Replatforming

AWS Mainframe Modernization Replatform with Rocket Software (formerly Micro Focus) provides a powerful replatforming option for businesses that want to migrate their mainframe applications to the cloud with minimal disruption. This solution enables you to recompile and run your existing COBOL and PL/I applications on AWS without requiring significant code changes.

Key benefits of the AWS Replatform with Rocket Software solution include:

- Preservation of existing business logic and investments
- Reduced risk and faster time to market
- Improved scalability and performance on AWS infrastructure
- Access to modern development tools and practices

You can use this solution to maintain your familiar mainframe programming languages while taking advantage of the flexibility, cost-effectiveness, and innovation of the AWS Cloud.

Automated refactoring

For a more transformative approach than replatforming, you can use AWS Blu Age, which offers automated refactoring of mainframe applications to Java-based cloud-native applications. This solution helps you modernize your legacy systems more comprehensively, and convert them into applications that can take full advantage of cloud-native technologies.

Key advantages of AWS Blu Age include:

- Conversion of legacy code to modern, maintainable Java applications

- Automated transformation that reduces manual effort and potential errors
- Creation of cloud-native applications that are optimized for AWS services
- Improved agility and easier integration with modern technologies

AWS Blu Age helps you migrate your applications and prepare them for the cloud, to open up new possibilities for innovation and growth. For more information about this approach, see Refactoring applications automatically with AWS Blu Age in the AWS Mainframe Modernization documentation.

Benefits of replatforming

This guide discusses an approach for replatforming mainframe COBOL applications on AWS. This approach aims to modernize legacy systems while temporarily retaining IBM Db2 for z/OS to streamline the transition process. By maintaining the existing database structure initially, you can reduce complexity and risk during migration. This phased approach helps you benefit from the scalability and cost-effectiveness of the AWS Cloud while preserving critical data integrity. The advantages of phased replatforming include the following:

- **Accelerated modernization:** Replatforming and refactoring typically require less time and resources compared with re-imagining a legacy application in the cloud, because they don't involve rewriting the entire application. This approach also supports a more gradual transition that enables organizations to modernize at their own pace while immediately benefiting from the scalability and cost-effectiveness of the AWS Cloud.
- **Risk mitigation:** Replatforming offers several advantages over refactoring for many organizations. Companies can maintain their existing COBOL and PL/I codebases, preserve years of business logic, and minimize the risk associated with large-scale code changes.
- **Data continuity and phased migration:** A significant benefit of replatforming is the option to initially keep data in Db2 for z/OS in its original data format. This strategy avoids the need for immediate, complex, and potentially risky data migration processes. By maintaining data in its original environment during the initial phase, you can preserve data integrity, reduce downtime, and minimize the risk of data loss or corruption during the modernization process. As a second step, you can plan for a controlled, phased data migration to cloud-native databases that involves thorough testing and validation while the application continues to run on the replatformed environment.
- **Flexibility and future-proofing:** For companies that have significant investments in mainframe skills and applications, replatforming provides a pragmatic path to modernization that balances

innovation with continuity. It offers the flexibility to retain critical data structures and access methods initially, while also setting the stage for future modernization efforts, including eventual data migration to fully cloud-native solutions.

Organizations can follow the replatforming approach to modernize at their own pace and address immediate needs while planning for long-term digital transformation goals. This approach also gives companies the opportunity to train their staff on cloud-native services.

The transformation process

Mainframe modernization is a critical step for organizations that want to leverage the benefits of cloud computing while preserving their valuable legacy applications. This transformation presents significant challenges. Mainframe applications are typically highly coupled and have intricate interdependencies that have evolved over decades of operation. This complexity necessitates a careful and methodical approach to modernization.

Organizations need to navigate the following key phases for a successful transition:

- **Planning**: This phase involves a comprehensive discovery of existing systems and prioritization of modernization efforts. Organizations assess their current infrastructure, identify critical applications, and determine which systems need to be modernized first.
- **Building**: During this stage, organizations create processes to migrate applications and develop new systems and infrastructure. This involves designing and implementing the modernized architecture and compiling the source code.
- **Running**: This step consists of creating the runtime environments to host the replatformed applications. It involves setting up the necessary hardware, software, and cloud infrastructure to support the modernized systems and to make sure that they can operate efficiently in the new environment.
- **Testing**: This phase includes rigorous validation of the modernized systems to verify that all functional and performance requirements have been met. Extensive testing is conducted to verify data integrity, system compatibility, and overall performance of the new environment.
- **Cutover**: The final phase focuses on implementing strategies for a smooth transition and controlling the shift from the legacy mainframe to the modernized environment. This includes careful planning of the migration schedule and contingency plans to minimize disruption to business operations.

The following sections discuss these phases in detail:

- Planning
- Building
- Running
- Testing
- Cutover

Planning

To navigate the mainframe legacy application's requirements effectively, organizations often begin with a comprehensive assessment of their mainframe environment.

Application discovery

A powerful tool in this initial phase is the Rocket Enterprise Analyzer, which provides deep insights into the structure, dependencies, and complexity of mainframe applications. This tool helps you determine the scope of your modernization effort, potential risks, and opportunities for optimization.

One crucial aspect to uncover is the intricate web of data dependencies within mainframe systems. These dependencies are often hidden beneath layers of legacy code and can significantly impact modernization efforts. By mapping out how different applications and modules interact with various data sources, you can better understand the potential effects of any changes you plan to implement.

Data dependencies

A thorough assessment of data dependencies can reveal critical information about data flow, data quality, and data governance within your mainframe environment. This knowledge is invaluable when planning data migration strategies, ensuring data integrity during modernization, and identifying opportunities for data optimization. By gaining a clear picture of your data, you can make more informed decisions about which modernization approaches will be most effective and least disruptive to your existing operations.

A top-down analysis that identifies the usage of tables by transactions or job control language (JCL) jobs is key to creating wave planning and prioritization. This approach clarifies the relationships between different components of your mainframe systems, and helps you develop a strategic, phased approach to modernization. By identifying which tables are most frequently accessed and by which processes, you can prioritize your modernization efforts: You can focus on high-impact areas first and ensure a smoother transition with minimal disruption to critical business operations.

In addition to using Rocket Enterprise Analyzer to discover data dependencies, many organizations also use their own custom-built solutions to gain deeper insights into their mainframe environments. These in-house tools often exploit the wealth of information that's available in the IBM Db2 catalog and System Management Facility (SMF) records.

Capacity benchmark

One step in planning your mainframe replatforming project is to gather detailed information about your current workload consumption. This data will help you accurately predict and provision the initial required capacity in your target cloud environment. For example, we recommend that you collect hourly million instructions per second (MIPS) consumption data for both online transactions and batch transactions from IBM Customer Information Control System (CICS) or Information Management System (IMS) and job control language (JCL) jobs.

IBM offers a diverse range of pricing models for MIPS in mainframe computing, and many of these models center around peak usage. Among these peak-based models, the most common is the rolling four-hour peak.

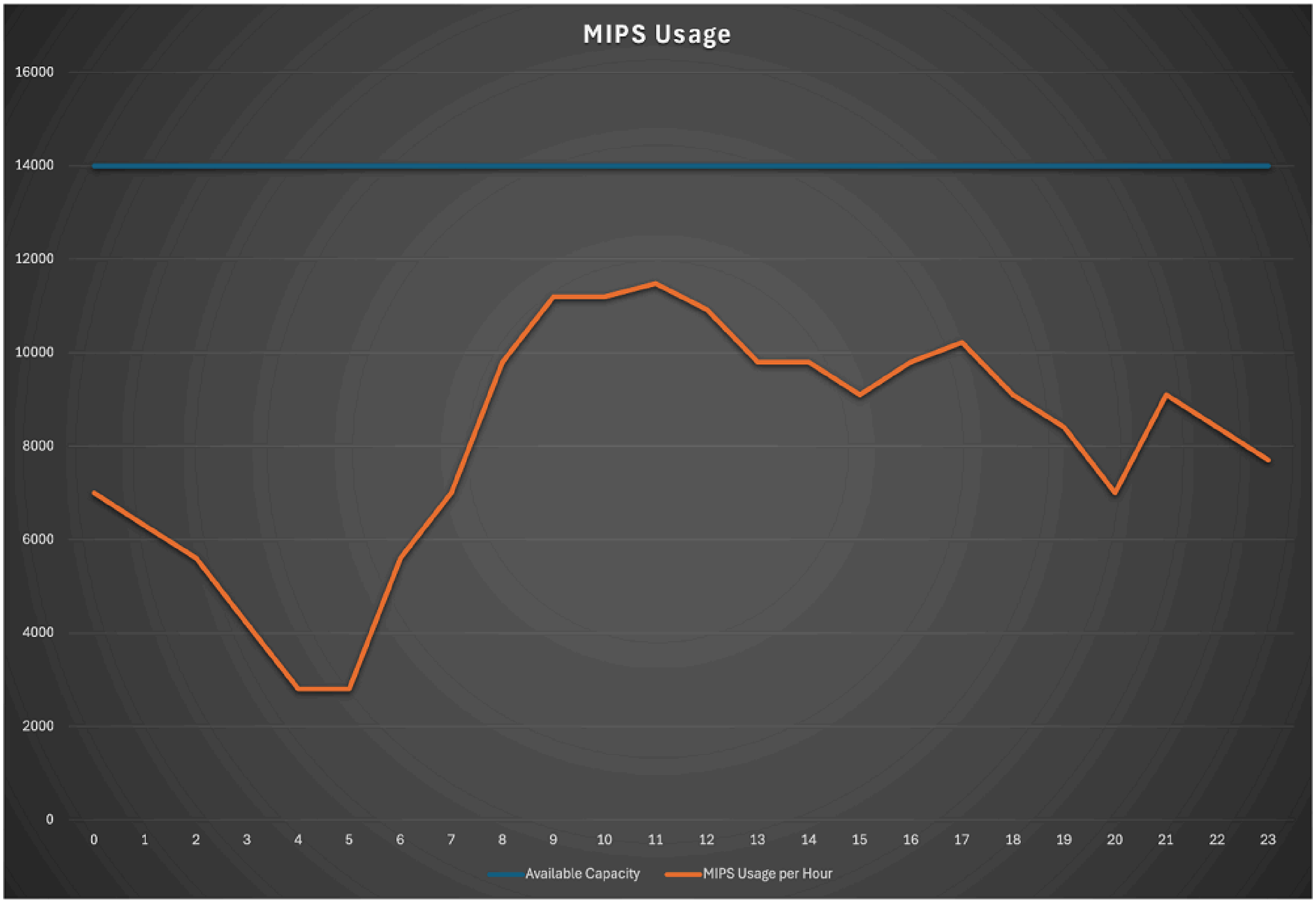
Mainframe costs include five key areas that significantly impact overall expenses:

- Software licensing is often a major component. It covers operating systems, middleware, databases, and various applications, and costs are sometimes tied to machine capacity or usage.
- Hardware expenses include the initial purchase or lease of mainframe equipment, ongoing maintenance, and upgrades.
- Storage costs can be substantial because of the vast amounts of managed data managed, and involve disk systems, tape libraries, and associated management software.
- Personnel expenses cover salaries for specialized mainframe professionals such as system programmers and database administrators.
- Disaster recovery and business continuity measures, including backup systems, redundant hardware, and offsite recovery facilities, represent a significant investment in ensuring high availability and quick recovery.

These five cost categories, combined with MIPS-based charges, form the core of most mainframe budgets. However, their relative proportions can vary widely depending on your organization's size, industry, and specific mainframe utilization patterns.

Hourly MIPS data is crucial for gaining a comprehensive understanding of your mainframe workload patterns and performance. Unlike daily or monthly averages, hourly data provides granular insights that reveal the nuanced fluctuations in your system's resource utilization throughout the day. This level of detail is invaluable for accurately assessing your application's performance and capacity needs in the cloud .

By analyzing hourly MIPS data, you can identify peak usage periods, spot trends, and pinpoint potential bottlenecks that might be obscured in aggregated data, as shown in the following diagram. This granularity allows for more precise capacity planning, helps optimize resource allocation, and can potentially lead to cost savings and improved system efficiency.



Hourly MIPS data also serves as an essential performance benchmark tool. It establishes a detailed baseline of your system's performance, which is particularly valuable when you're planning or evaluating system changes such as migrations or upgrades. By comparing pre-change and post-change hourly MIPS data, you can accurately measure the impact of these modifications on your system's performance and ensure that your mainframe continues to meet your organization's needs.

To collect hourly MIPS data, you have several options. One approach is to use SMF records directly. These records provide a wealth of information about system activity and resource usage. Alternatively, you can use specialized tools such as the IBM Sub-Capacity Reporting Tool (SCRT), which can simplify the process of collecting and analyzing MIPS data.

Regardless of the method you choose, it's important to collect data over an extended period —ideally, several months. This extended collection period enables you to account for cyclical variations in your workload, such as end-of-month processing spikes or seasonal fluctuations. By capturing these long-term patterns, you can develop a more accurate and comprehensive picture of your mainframe's performance characteristics, which enables better-informed decision-making and more effective capacity management.

Wave planning

You can use the information you gather to strategically prioritize your mainframe replatforming initiatives. A prudent approach is to begin with less critical workloads, such as non-core business transactions or batch jobs, to allow teams to gain experience and refine processes with minimal risk to essential operations. Additionally, considering read-only workloads as early candidates for migration can be advantageous, because these workloads typically involve less complexity and lower risk of data inconsistencies. This approach enables you to build confidence and momentum in your replatforming efforts.

In addition, grouping workloads that share Db2 tables for write or update operations can streamline the migration process. By identifying these interconnected workloads, you can plan cohesive migration waves that maintain data integrity and minimize the need for complex interim solutions. This strategy not only reduces the risk of data conflicts but also optimizes the overall replatforming timeline by addressing related components simultaneously. Ultimately, this data-driven prioritization approach ensures a balanced consideration of criticality, complexity, and interdependence, and leads to a more efficient and successful mainframe modernization process.

Building

Using a shared Db2 database enables concurrent execution of identical or consistent applications in both mainframe and cloud environments. This approach offers several advantages when you maintain the same application version across both platforms, and provides enhanced flexibility and reliability in your operations.

One key advantage of this strategy is the ability to implement an effective rollback plan. If issues arise during migration or deployment, having the same application version allows for a seamless reversion to the previous state, and minimizes downtime and potential data inconsistencies.

Application consistency

Mirroring application components from a distributed source control manager to the mainframe is a strategic approach during the replatforming process. This method supports the use of modern source code management tools while maintaining synchronization with the mainframe environment. This mirroring process is temporary, and lasts only until the workload is fully functional in production on the distributed platform.

By migrating your replatformed application's source code to a distributed change management tool, you can take advantage of several benefits offered by modern source code managers. These include:

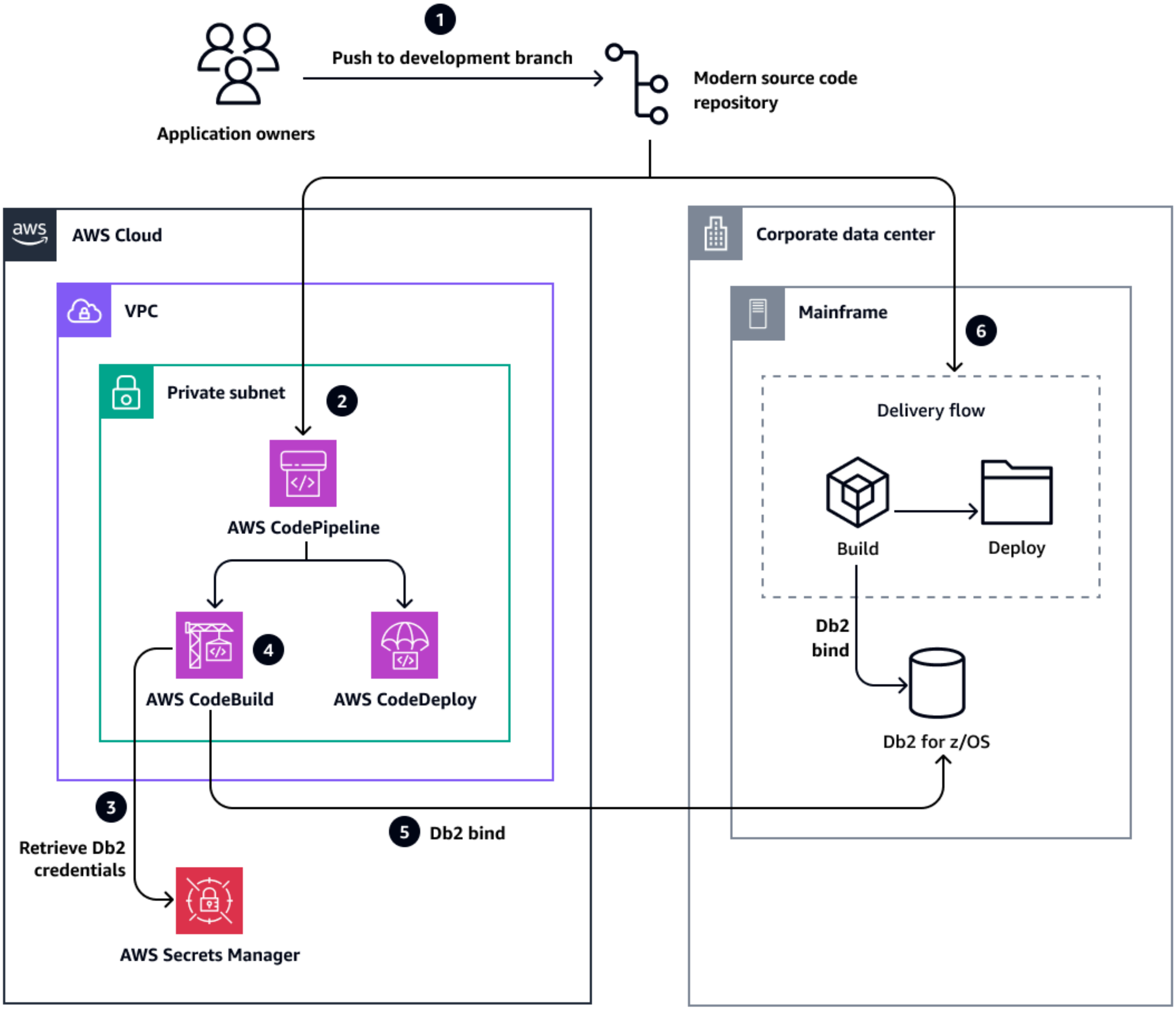
- **Enhanced collaboration**: Distributed tools often provide better support for team collaboration by including features such as pull requests, code reviews, and branching strategies.
- **Improved version control**: Modern systems offer more granular version control, and make it easier to track changes and manage different versions of the code.
- **Integration with CI/CD pipelines**: Many distributed tools seamlessly integrate with continuous integration and continuous deployment (CI/CD) pipelines, which streamline the development process.
- **Better visibility and traceability**: These tools often provide superior dashboards and reporting capabilities, and offer greater insight into the development process.
- **Support for modern development practices**: Distributed systems are typically better suited for agile methodologies and DevOps practices.

The mirroring process involves synchronizing the code from the distributed source control manager back to the mainframe. This ensures that both environments remain consistent during the transition period. However, you must implement mirroring as a one-way synchronization, where updates flow from the distributed system to the mainframe, instead of bidirectionally. This approach maintains consistency and prevents potential conflicts that could arise from simultaneous updates in both environments.

By adopting this mirroring strategy, you can gradually shift your development efforts to the distributed platform while ensuring that the mainframe environment remains up-to-date. This provides a smoother transition and a safety net during the replatforming process. When the workload is fully functional and stable in the distributed production environment, you can phase out the mirroring process and complete the migration to the modern source code management system.

Architecture

The following diagram shows how a distributed source code management system can mirror application components and maintain synchronization between the AWS Cloud and mainframe environments. The AWS Cloud environment uses CI/CD services such as AWS CodeBuild, AWS CodePipeline, and AWS CodeDeploy to build and deploy the application.



In this workflow:

1. The application owners deliver a new application release into the development branch of the source code repository.
2. The new release triggers AWS CodePipeline.
3. AWS CodeBuild retrieves Db2 credentials from AWS Secrets Manager.
4. CodeBuild compiles the application.
5. CodeBuild uses Db2 for z/OS to bind the application.
6. The mainframe delivery flow builds and deploys the application as well.

Running

To ensure optimal performance and low latency between your cloud-based application and your on-premises database, we recommend that you implement AWS Direct Connect. This service provides a dedicated network connection between AWS and your organization's data center, and offers more consistent network performance and reduced latency compared with internet-based connections. This is particularly crucial for database operations that require quick response times.

To achieve high availability (HA) and elasticity for the application that's running on AWS, you can implement a robust architecture by using the following components:

- **Elastic Load Balancing (ELB)**: You can deploy a load balancer to distribute incoming traffic across multiple Amazon Elastic Compute Cloud (Amazon EC2) instances that your application runs on. This ensures even distribution of the workload and provides a single entry point for client requests.
- **Auto Scaling group**: EC2 instances that host the application can be organized into an Auto Scaling group. This allows the infrastructure to automatically adjust the number of instances based on predefined metrics such as CPU utilization or network traffic. During peak times, additional instances can be launched to handle increased load, whereas during quieter periods, unnecessary instances can be terminated to optimize costs.
- **EC2 instances**: The application can be deployed on EC2 instances within the Auto Scaling group. These instances should be distributed across multiple Availability Zones to enhance fault tolerance and ensure high availability.
- **Multi-AZ deployment**: By spreading the application instances across multiple Availability Zones, the system can withstand the failure of a single Availability Zone without significant impact on overall availability.

This architecture enables the application to scale seamlessly based on demand while maintaining high availability. The load balancer ensures that traffic is distributed evenly across healthy instances, and the Auto Scaling group manages the number of instances based on actual workload.

To further enhance reliability, you can implement a robust monitoring and alerting system by using Amazon CloudWatch to help detect and respond to any performance issues or failures promptly. Additionally, regular testing of the automatic scaling capabilities and failover scenarios will ensure that the system behaves as expected during various load conditions and potential failures.

By adopting this approach, you can benefit from the scalability and flexibility of the AWS Cloud while maintaining a secure connection to your on-premises Db2 database. This hybrid setup serves as an excellent path toward a full cloud migration, and provides gradual transition and risk mitigation throughout the process.

Two-phase commit (2PC)

AWS Mainframe Modernization Replatform with Rocket Software offers support for two-phase commit (2PC) transactions through its implementation of extended architecture (XA). This capability is crucial for maintaining data integrity across distributed systems, particularly in mainframe environments where complex transactions often span multiple resources.

The XA architecture, which is integrated into AWS Replatform with Rocket Software, enables the coordination of transactions across diverse resources such as databases and message queues. This integration ensures that all parts of a distributed transaction either commit or roll back in unison, to maintain consistency across the system.

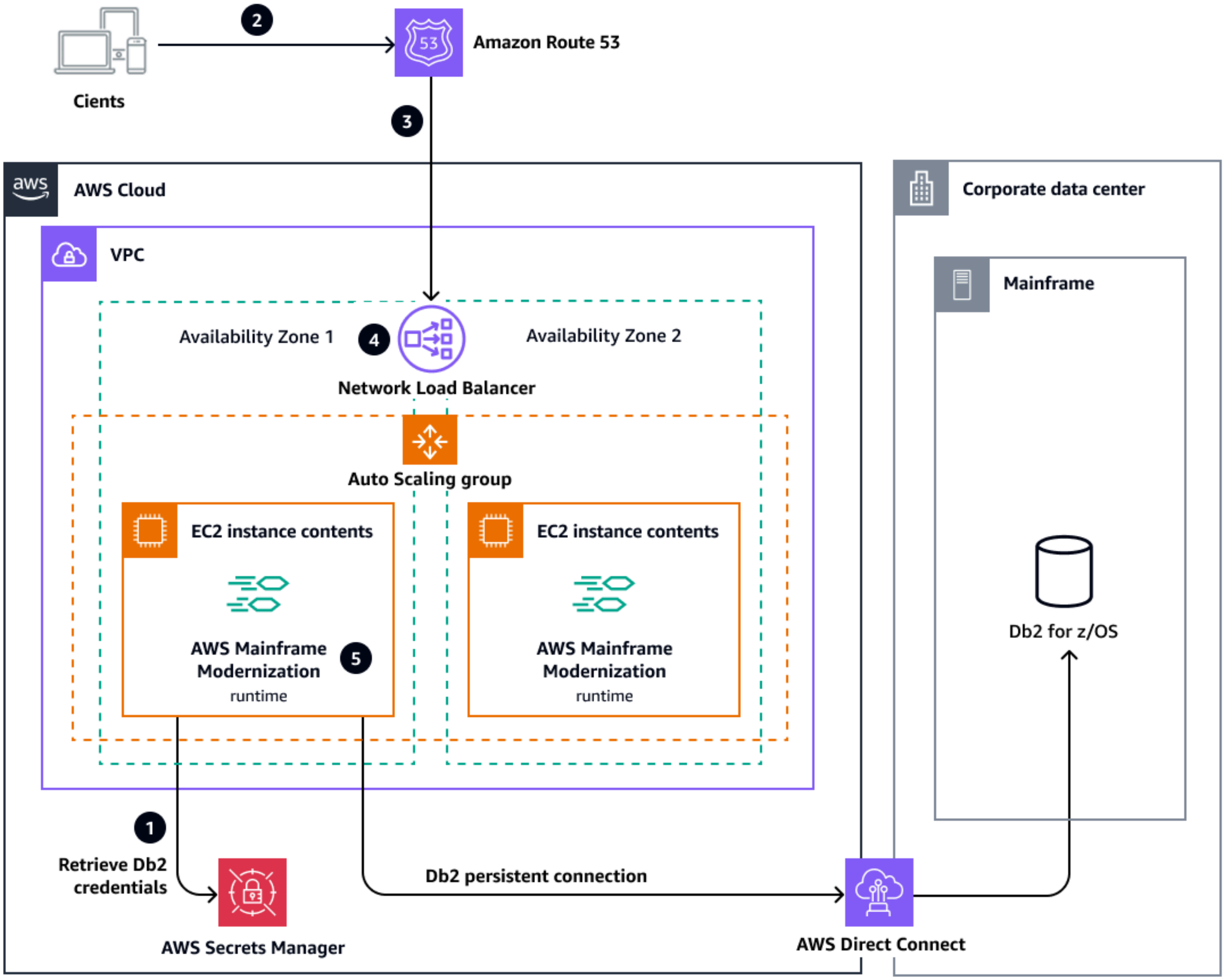
The 2PC process consists of two phases:

- Prepare phase: The transaction manager queries all resource managers involved in the transaction to ensure that they're ready to commit.
- Commit phase: If all resource managers respond positively, the transaction manager instructs them to commit the changes. If any of the resource managers cannot commit, all managers are instructed to roll back changes.

By using XA, AWS Replatform with Rocket Software provides a reliable and scalable solution for managing complex, distributed transactions in modernized mainframe environments. This feature is essential for organizations that want to migrate their mainframe applications to the cloud without compromising on transactional integrity or performance.

Runtime infrastructure

The following diagram shows a highly available and elastic environment in the AWS Cloud that includes two Availability Zones, EC2 instances in an Auto Scaling group, a Network Load Balancer, and a dedicated connection between the AWS and mainframe environments through AWS Direct Connect.



In this architecture:

1. When the AWS Mainframe Modernization runtime starts, it retrieves Db2 credentials from AWS Secrets Manager and opens a persistent connection with Db2 for z/OS.
2. Clients bind the Network Load Balancer address in Amazon Route 53.

3. Route 53 redirects transactions to the the Network Load Balancer.
4. The Network Load Balancer distributes transactions across multiple EC2 instances.
5. The workload that's running on AWS Mainframe Modernization interacts with Db2 for z/OS by using a persistent connection through AWS Direct Connect.

Testing

When you replatform a COBOL application while maintaining Db2 for z/OS as the shared database, it's crucial to ensure that the new system functions equivalently to the original. This hybrid environment presents unique challenges and opportunities for testing. The following strategy outlines a comprehensive approach to functional equivalence testing and is designed to validate the replatformed application's performance, data integrity, and seamless integration with the existing Db2 for z/OS database.

Start by identifying the critical business processes and transactions that need to be compared between systems. Then, create a detailed test plan with specific scenarios that will effectively evaluate the functional equivalence of these transactions. Finally, develop comprehensive test data sets that cover all identified scenarios, and make sure that they are identical for both systems to enable accurate comparison.

Source environment

- Initial snapshot (first snapshot):
  - Make sure that the data table isn't being used by other applications during the test, because this can affect the equivalence test.
  - Take a snapshot of the Db2 for z/OS tables that are used by the transaction before running any tests.
- Source system testing:
  - Run the full suite of tests on the original COBOL application.
  - Record all transactions, inputs, and outputs.
  - Monitor system performance and resource utilization.
- Post-source testing snapshot (second snapshot):
  - Take another snapshot of the Db2 for z/OS database after you complete the source system tests.

Target environment

- Database reset:
  - Restore the database to its initial state by using the first snapshot.
- Target system testing (replatformed environment):
  - Run the same suite of tests on the replatformed application.
  - Make sure that all target system tests use the same inputs as the source system tests.
  - Monitor system performance and resource utilization.
- Post-target testing snapshot (third snapshot):
  - Take a final snapshot of the Db2 for z/OS database after you complete the target system tests.

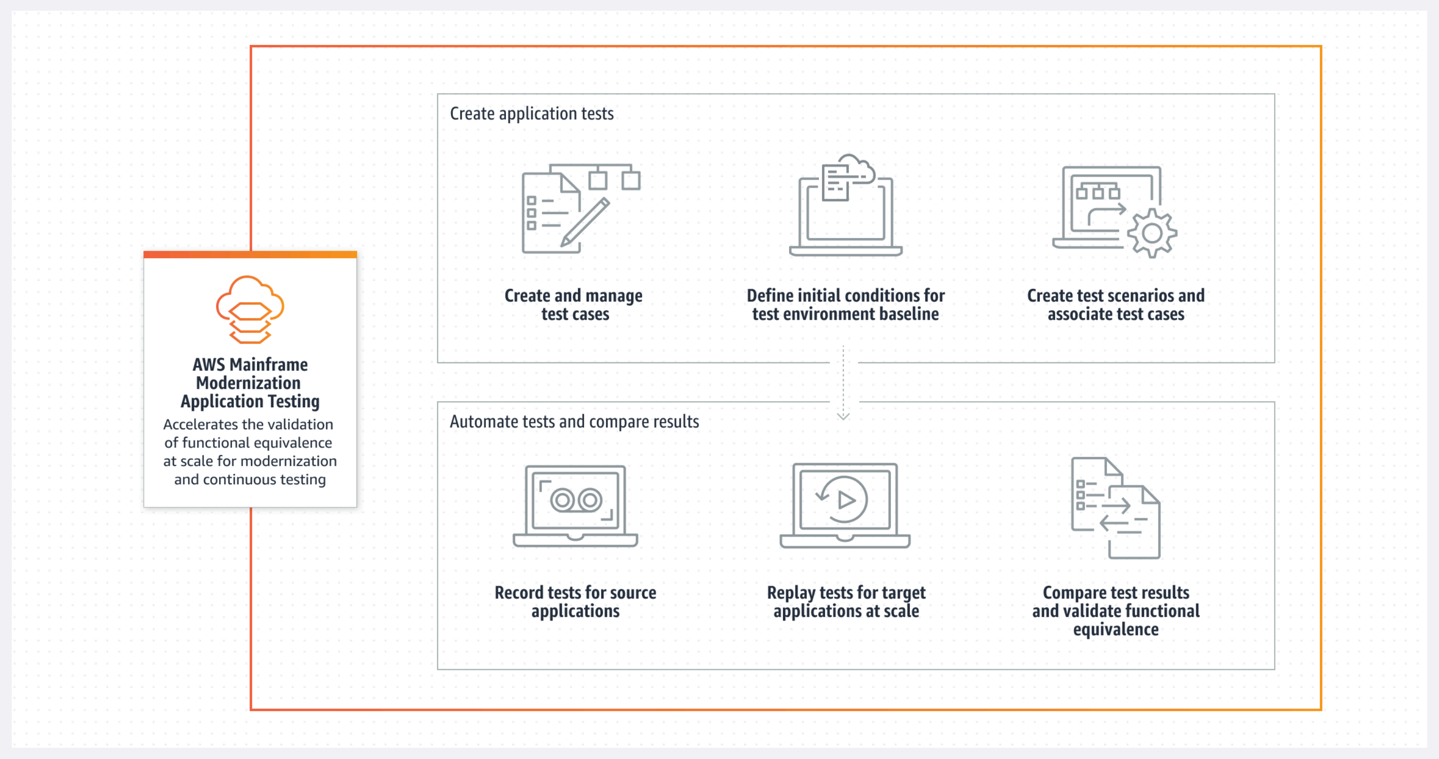
Analysis

- Comparison and analysis:
  - Compare the second and third snapshots to identify any discrepancies in data.
  - Analyze test results, and compare the outputs from the source and target systems.
  - Evaluate performance metrics between the two environments.
- Integration testing:
  - Perform tests that involve both the replatformed application and any remaining COBOL components.
  - Verify seamless interaction between the two environments.
- Failover and recovery testing:
  - Test scenarios where one environment fails and the other environment takes over.
  - Ensure data consistency and integrity during failover situations.
- Load and stress testing:
  - Conduct tests with varying loads to assess how the hybrid system performs under stress.
  - Identify any bottlenecks or performance issues in either environment.
- Documentation and reporting:
  - Document all test results, discrepancies, and performance metrics.
  - Prepare a comprehensive report that compares the source and target systems.

Testing your application in AWS Mainframe Modernization

AWS offers the AWS Mainframe Modernization Application Testing service to automate the execution of application tests at scale. AWS Application Testing helps optimize and reduce mainframe application modernization and testing project costs.

The following diagram shows how AWS Application Testing works at a high level.


AWS Mainframe Modernization Application Testing
Accelerates the validation of functional equivalence at scale for modernization and continuous testing
Create application tests
Create and manage test cases
Define initial conditions for test environment baseline
Create test scenarios and associate test cases
Automate tests and compare results
Record tests for source applications
Replay tests for target applications at scale
Compare test results and validate functional equivalence

The process consists of these steps:

1. Create and manage test cases, which are the smallest unit of testing actions. Identify the data types that best represent functional equivalence between the source and target systems.
2. Define the configuration of the test environment by specifying AWS CloudFormation templates and additional attributes.
3. Create test suites, which are collections of test cases.
4. Upload and replay data sets: Capture the input and output data sets on the mainframe, upload them to AWS, and then replay the test scenario on the target system.
5. Compare source and target data sets. AWS Application Testing automatically compares the output data sets from the source and target systems. Review and evaluate these to identify discrepancies.

For more information, see the AWS Mainframe Modernization documentation.

Cutover

In mainframe modernization, one of the most critical challenges is minimizing downtime and risk during the transition to a new platform. The blue/green deployment strategy offers a powerful and flexible approach to system migration.

Blue/green deployment is a technique that reduces downtime and risk by running two identical production environments called *blue* and *green*. Here's how it works in the mainframe modernization context:

- Blue environment: This is your current mainframe system that's handling all the production traffic.
- Green environment: This is your new, modernized platform on AWS that's ready to take over.

The blue/green cutover strategy includes these steps: provision, go live, roll back if issues arise, and conclude.

Provision

In this stage, you provision the new (green) environment on AWS by following these steps:

1. **Replatform the environment**: The Route 53 hosted zone must contain a DNS record that points to the mainframe environment (blue).
2. **Verify connectivity:** Ensure proper connection between your AWS account and on-premises transaction managers and Db2 for z/OS database.
3. **Run smoke tests:** Use the AWS load balancer address to access the replatformed environment and perform comprehensive smoke tests to verify the following:
   - All expected workloads are available.
   - 3270 transactions are processing correctly.
   - Data interactions with Db2 for z/OS are functioning as expected.

Go live

In this stage, you shift the traffic to the green environment and monitor the changes.

1. Use the traffic routing policies in Route 53 to shift traffic:
   - Option A: You can shift traffic all at once.
   - Option B: Alternatively, you can use a gradual weighted distribution.
2. Monitor and validate:
   - Closely watch the AWS environment as traffic shifts.
   - Check 3270 transaction processing.
   - Verify Db2 for z/OS communication.
   - Monitor for performance issues.
   - Have users validate transaction results.

Rollback

If issues arise, you can quickly update Route 53 to redirect traffic back to the on-premises mainframe (blue) environment.

You should investigate and resolve problems before you attempt another cutover.

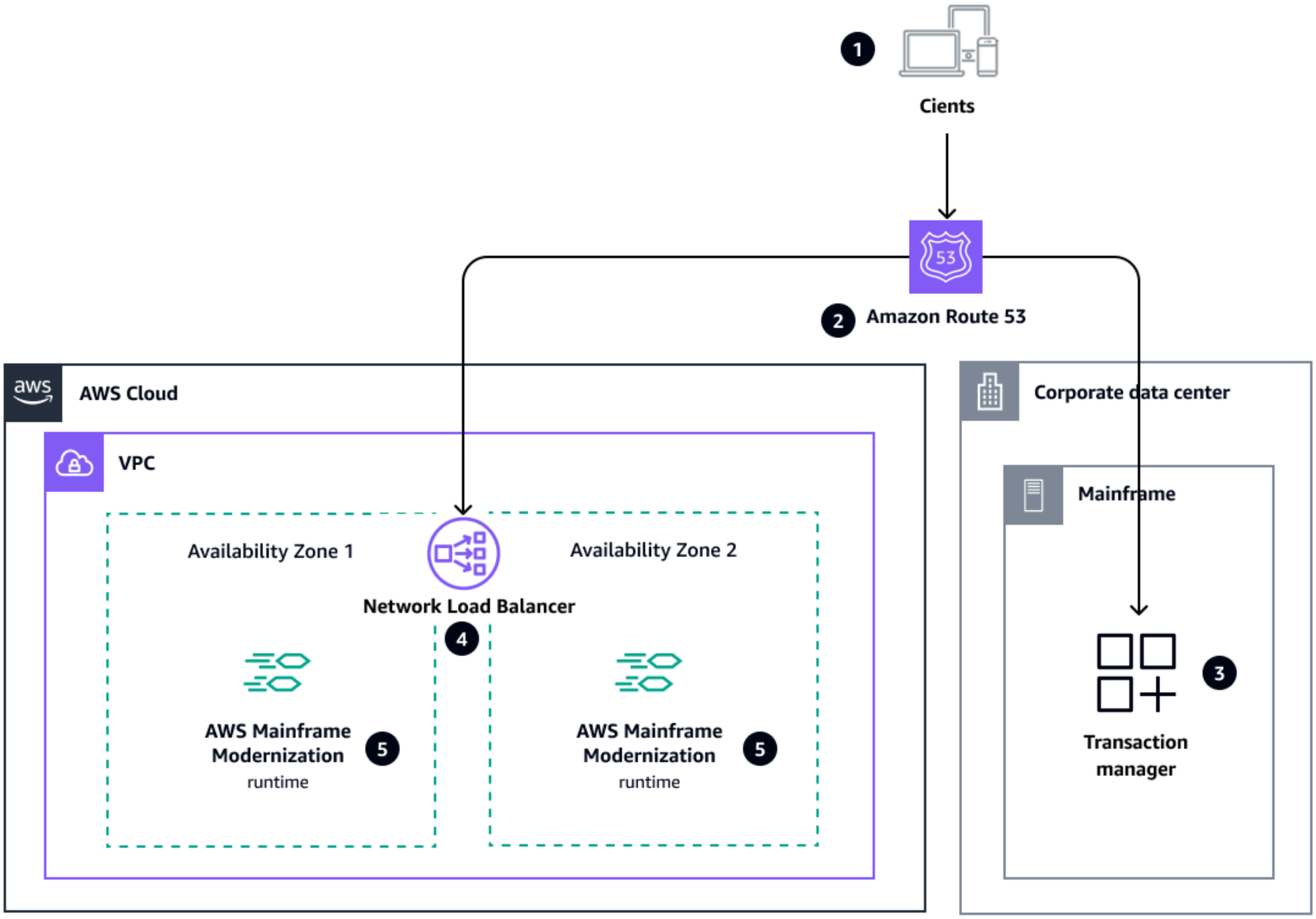
Conclude

After you've monitored the traffic and validated that your green environment is functioning correctly, you can gradually increase the application traffic to AWS.

After a stable period, you can decommission the mainframe transaction (blue) environment, and keep the Db2 for z/OS database on premises.

Architecture

The following diagram illustrates the cutover flow.



The cutover process consists of the following:

1. The client applications, frontends, and backends for frontends (BFFs) send transactions to the Route 53 domain name.
2. Route 53 routes the connection to the mainframe transaction manager or to the Network Load Balancer, depending on the defined routing policy.
3. The transaction manager processes the transactions that are sent to the mainframe.
4. The Network Load Balancer distributes transactions to the available replatform environments for processing.
5. The AWS Mainframe Modernization replatform environments process the requests.

Best practices

This section outlines a set of best practices for addressing key challenges in replatforming mainframe workloads to cloud environments while keeping the database on Db2 for z/OS.

Network latency

To accurately predict the latency impact of separating the application from the Db2 database during a replatforming effort, we recommend that you conduct a thorough evaluation of the number of Db2 calls for both transactions and batch processes. This evaluation should be done by using trace data and should include the following steps:

- **Collect trace data**: Gather detailed traces of representative transactions and batch jobs, and make sure that the traces capture all Db2 interactions, including entries and exits.
- **Analyze the trace data**: Count the number of Db2 entries and exits for each transaction and batch job, and calculate the average number of Db2 interactions per transaction and batch process.
- **Measure current response times**: Check if Db2 access is aligned with your application's service-level agreement (SLA).
- **Estimate network latency**: Determine the expected network latency between the replatformed application and the Db2 database. Consider factors such as physical distance, network infrastructure, and potential bottlenecks.
- **Calculate potential impact**: For each transaction and batch process, multiply the number of Db2 entries and exits by the estimated network latency. Add this calculated time to the current response times to predict the new total processing time.
- **Assess the results**: Evaluate whether the predicted latency increase is acceptable for your business requirements and identify any transactions or processes that might require optimization or redesign to mitigate latency issues.
- **Consider mitigation strategies**: Explore options such as connection pooling, caching, or batch data retrieval to reduce the number of individual Db2 interactions. Evaluate the possibility of moving frequently accessed data closer to the application tier.

By following these steps, you'll be able to make data-driven decisions about the feasibility of your replatforming strategy and identify any potential performance issues before they impact your production environment. This approach will help ensure a smooth transition while maintaining acceptable performance levels for your database-dependent applications.

Security

- **Secure your application build**: Use a private subnet in the virtual private cloud (VPC) to run AWS CodeBuild to help ensure isolation and enhanced security. Implement a Db2 trusted context from the CodeBuild subnet CIDR for secure database access during the build process.
- **Secure your runtime environment**: Use a Db2 trusted context from the runtime subnet CIDR for secure database connections.
- **Manage database credentials securely:** Implement a regular credential rotation schedule to minimize the risk of unauthorized access. Store Db2 credentials securely in AWS Secrets Manager.
- **Establish network security**: Implement strong network segmentation and firewall rules to protect both build and runtime environments. Use the correct combination of AWS Direct Connect and AWS Site-to-Site VPN to achieve the necessary level of security.
- **Enforce encryption**: Enforce encryption for data in transit between your application and Db2 for z/OS.

Application governance

- **Establish a source of truth**: Establish the new software configuration management (SCM)—for example, GitHub—as the single source of truth for the migrated application code. This ensures consistency and eliminates version discrepancies between the cloud and mainframe environments during the transition period.
- **Update the change management process**: Update the change management process to consider code modifications in this new, dual-environment paradigm. This process should include:
  - Clear approval workflows for code changes.
  - Mandatory code reviews before merging code into the main branch.
  - Synchronized deployment procedures to ensure that both environments receive updates simultaneously.
  - Rollback mechanisms in case of issues in either environment.

Elasticity

The elasticity of cloud computing introduces a paradigm shift that significantly alters the mainframe cost structure and resource management. Unlike the traditional mainframe environment, which has fixed capacity and peak-based pricing models, cloud platforms offer dynamic scalability and a pay-as-you-go approach that can potentially lead to substantial cost savings and improved operational efficiency.

In a cloud environment, organizations can scale their computing resources up or down in real time based on actual demand, which eliminates the need for overprovisioning to accommodate peak loads. This elasticity allows businesses to pay only for the resources they consume instead of investing in expensive hardware and software licenses to handle occasional spikes in usage.

For details on how pricing works on AWS, see AWS Pricing.

Next steps

Mainframe modernization is a complex and critical initiative that demands specialized knowledge and advanced solutions. You can accelerate your modernization process and achieve faster business outcomes through strategic partnerships that will help you with the following tasks:

- **Evaluate and prioritize**: Review your mainframe applications and identify which ones are suitable for replatforming while keeping the database on Db2 for z/OS. Consider factors such as complexity, business criticality, and potential return on investment (ROI).
- **Develop a migration strategy**: Create a detailed plan for replatforming your selected applications, including timelines, resource allocation, and risk mitigation strategies.
- **Assess tools and technologies**: Research and select appropriate tools and technologies to facilitate the replatforming process, such as application modernization platforms or code conversion tools.
- **Engage with experts**: Consider partnering with mainframe modernization specialists or consulting firms that have experience in replatforming projects.
- **Proof of concept**: Start with a small-scale proof of concept to validate your approach and identify potential challenges before scaling up to larger applications.
- **Testing and validation**: Develop a comprehensive testing strategy to ensure that your replatformed applications function correctly and maintain data integrity with your existing Db2 for z/OS database.
- **Training and knowledge transfer**: Prepare your team for the new environment by providing training on the replatformed applications and any new tools or technologies introduced.
- **Phased implementation**: Consider a phased approach to replatforming, where you gradually migrate applications while monitoring performance and addressing any issues that arise.
- **Continuous optimization**: After replatforming, continuously monitor and optimize the performance of your applications and their interactions with the Db2 for z/OS database to ensure long-term success.
- **Modernize at your pace**: Now that the workload is running on AWS and already taking advantage of the cloud, start planning the re-imagine phase of your modernization.

Resources

For more information about mainframe migration and modernization, see the following resources.

AWS documentation

- Configuring Amazon Route 53 as your DNS service
- Routing traffic to an ELB load balancer
- Weighted routing
- Replatforming applications with Rocket Software

Rocket Software references

- Micro Focus External Call Interface (ECI)
- CICS Web Services

IBM references

- Trusted contexts (IBM Db2 for z/OS documentation)

Tools

- Rocket Enterprise Server

AWS Prescriptive Guidance patterns and guides

- Build COBOL Db2 programs by using AWS Mainframe Modernization and AWS CodeBuild
- DevOps for AWS Mainframe Modernization
- Mainframe modernization: Decoupling patterns for migrating application code
- Secure and streamline user access in a Db2 federation database on AWS by using trusted contexts

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	May 7, 2025

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model’s predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application’s environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic `PUT` and `DELETE` requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the

AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don’t know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn’t provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.