AWS Prescriptive Guidance

Portfolio playbook for AWS large migrations

AWS Prescriptive Guidance: Portfolio playbook for AWS large migrations

Table of Contents

Portfolio playbook for AWS large migrations

AWS Professional Services

February 2022

In a large migration, the portfolio workstream plans waves of applications for migration, and the migration workstream focuses on migrating those waves. When planning waves, the portfolio workstream is responsible for assessing the portfolio, collecting the metadata needed for the migration, prioritizing the applications, and then assigning the applications to waves. Waves must be sized and scheduled according to the capacity of the migration workstream and must account for the complexity of the application, dependencies, and any business factors, such as budgets, performance goals, resource availability, and deadlines.

In the first stage, initializing a large migration, you create the runbooks that the portfolio workstream uses to assess the portfolio and plan waves. In the second stage, implementing a large migration, the portfolio workstream uses those runbooks to assess the portfolio and plan waves and refines the runbooks as they repeat the processes in them again and again.

For more information about core and supporting workstreams, see Workstreams in a large migration in the *Foundation playbook for AWS large migrations.*

About this playbook

This guide is part of a series about large migrations to the Amazon Web Services (AWS) Cloud. Migrating 300 or more servers is considered a large migration. If you haven't already done so, we highly recommend reading the following in the order listed:

1. AWS large-migration strategy and best practices – This strategy discusses best practices for large migrations and provides use cases from customers across various industries.
2. Guide for AWS large migrations – This guide describes a high-level, phased approach to implementing the best practices outlined in the strategy document.
3. Application portfolio assessment guide for AWS Cloud migration – This guide helps you complete a high-level assessment of the application portfolio and associated infrastructure.

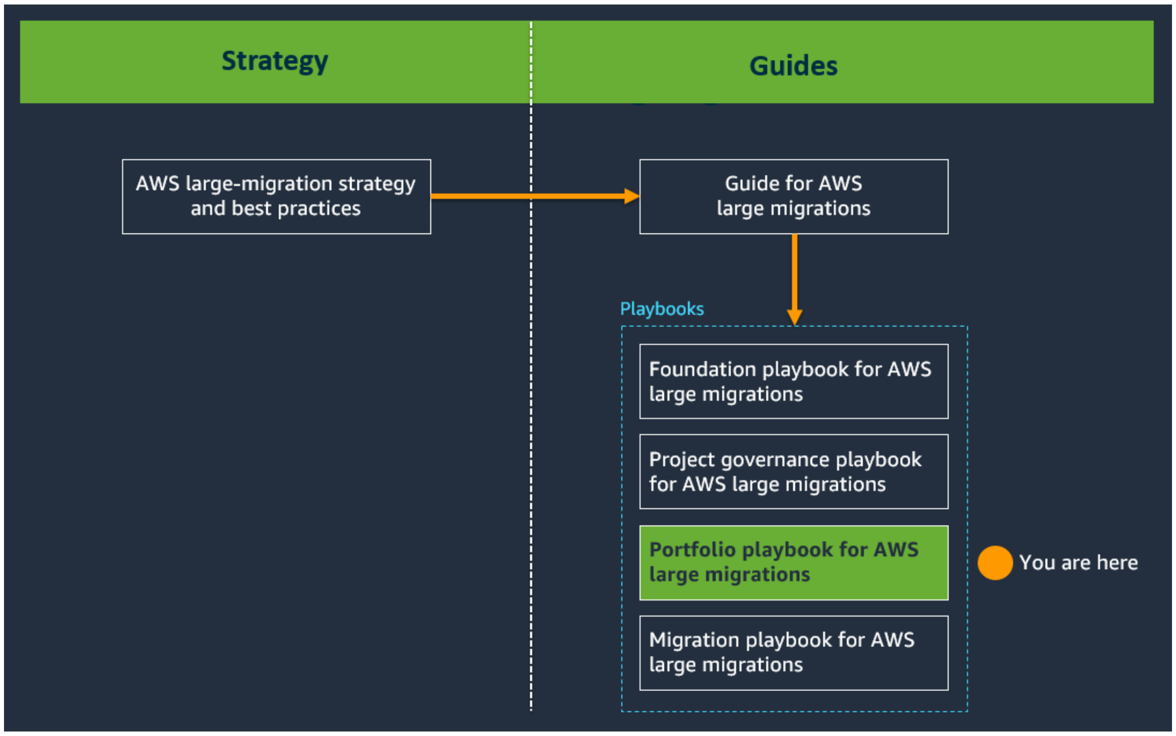
    Note
    Performing an initial, high-level discovery and assessment of the application portfolio is a prerequisite to completing the tasks in this playbook.

After reading the strategy and guides, the playbooks help you focus on how to implement the strategy and connect all the moving parts of a large migration. A *playbook* helps you build the assets and processes that you use in a large migration, such as runbooks, governing documents, or tools. A *runbook* is a standard operating procedure for a task that you perform repeatedly. The following playbooks are available in this document series for large migrations, and we recommend that you read them in the following order:

1. Foundation playbook for AWS large migrations
2. Project governance playbook for AWS large migrations

3. Portfolio playbook for AWS large migrations (this guide)
4. Migration playbook for AWS large migrations

The following figure shows the structure of the AWS documentation series for large migrations. Review the strategy first, then the guide, and then read the playbooks.


Strategy
Guides
AWS large-migration strategy and best practices
Guide for AWS large migrations
Playbooks
Foundation playbook for AWS large migrations
Project governance playbook for AWS large migrations
Portfolio playbook for AWS large migrations
Migration playbook for AWS large migrations
You are here

This playbook provides a step-by-step approach to performing a detailed portfolio assessment for a large migration project, including application assessment and wave planning. It describes the tasks of the portfolio workstream, which spans both stages of a large migration, initialization and implementation:

- In stage 1, *initialize*, you validate your initial portfolio discovery and migration strategy, and you create runbooks that define the processes and rules used for portfolio assessment and wave planning. At the end of stage 1, you have portfolio runbooks and tracking tools that are customized for your own portfolio, processes, and infrastructure.
- In stage 2, *implement*, you use the runbooks you created in the previous stage in order to complete the portfolio assessment and wave plans.

Detailed portfolio assessment and wave planning is not a one-off task. It is a continuous workstream that supports the migration. In a migration factory, portfolio assessment and wave planning provide the raw materials (servers) to the factory, so you must continue with these activities until the migration project is complete. For more information about the migration factory model, see the Guide for AWS large migrations.

About the runbooks, tools, and templates

In this playbook, you create the following runbooks:

- Application prioritization runbook
- Metadata management runbook

- Wave planning runbook

In addition, you create the following tools, which you use for tracking progress or documenting decisions and other important information:

- Application complexity score sheet
- Application target state worksheet
- Portfolio assessment progress tracker
- Questionnaire for application owners
- Wave planning and migration dashboard

We recommend using the portfolio playbook templates and then customizing them for your portfolio, processes, and environment. The instructions in this playbook tell you when and how to customize each of these templates. This playbook includes the following templates:

- **Application target state worksheet** – You use this template to define the future state of an application in the AWS Cloud when the application or migration strategy is particularly complex.
- **Dashboard template for wave planning and migration** – You use this template to collate critical metadata, analyze the application portfolio, identify dependencies, and plan the migration waves.
- **Progress tracking template for portfolio assessment** – You use this template to track the progress of each application through the portfolio workstream.
- **Questionnaire template for application owners** – You use this template in the application deep dive process in order to collect information about the application directly from the application owners.
- **Runbook template for application prioritization** – This template is a starting point for building your own application prioritization and deep dive processes.
- **Runbook template for metadata management** – This template is a starting point for building your own metadata identification and collection processes.
- **Runbook template for wave planning** – This template is a starting point for building your own wave planning processes.
- **Score sheet template for application complexity** – You can use this template to evaluate the complexity of migrating each application to the cloud, and then you can use the resulting score during the application prioritization process.

Stage 1: Initializing a large migration

In the initialize stage, you define the runbooks that you use to complete a detailed portfolio assessment and wave plan in the implementation stage. If another team member is responsible for defining the runbooks in your large migration project, skip to Stage 2: Implementing a large migration (p. 49), where you will use the runbooks to migrate waves of applications and servers. By documenting the decisions made in this stage, you create actionable runbooks. For example, making decisions about the following questions leads to standard procedures that you document in your portfolio runbooks:

- What migration metadata is required, and how do you collect it?
- How do you prioritize applications and perform deep dives?
- How do you plan waves?

In stage 1, you spend a lot of time defining the rules and building the runbooks because the activities in the runbook are repeated many times in stage 2 to support the migration.

Stage 1 consists of the following tasks and steps

Task 1: Performing the initial discovery and validating the migration strategy

The first step of portfolio assessment in a large migration project is to understand the information you have today, the business and technical drivers, and any migration strategy decisions that have already been made. The outcome of the portfolio assessment is to continuously feed the migration metadata, wave plan, and migration strategies into the migration workstream. Based on the information collected, you analyze the gaps and decide the next steps. You can skip some of the sections in this playbook if you have completed the analysis and tasks already. This task consists of the following steps:

- Step 1: Validate the discovery data (p. 5)
- Step 2: Identify the business and technical drivers (p. 6)
- Step 3: Validate the migration strategies (p. 7)
- Step 4: Validate the migration patterns (p. 9)

Step 1: Validate the discovery data

In the mobilize phase, you might have completed your initial portfolio assessment, and if so, you can reuse that discovery data in the migrate phase. If not, don't worry. This playbook will walk you through what is needed to support your large migration.

Large migrations usually have a lot of data. For example, you have:

- Metadata about the source servers, applications, and databases
- Information about your IT portfolio from your configuration management database (CMDB)
- Data from discovery tools that helps you better understand the current state and dependencies
- Metadata for target AWS resources

About the types of metadata

The following are the three primary types of metadata that are required to support a large migration:

- **Source portfolio metadata** – Source portfolio metadata is the metadata about your source servers, applications, and databases. You can get the metadata from an existing CMDB, discovery tools, or even from the application owner. You can find a comprehensive list of this metadata type here, and the following are some examples:
  - Server name
  - Server IP address
  - Server operating system (OS)
  - Server storage, CPU, memory, and input/output operations per second (IOPS)
  - Application name
  - Application owner
  - Application-to-application dependencies
  - Business unit
  - Application-to-server mapping
  - Application-to-database mapping
  - Database type and size
  - Storage type and size
  - Dependencies metadata
  - Performance and usage data
- **Target environment metadata** – This is a metadata type that helps you migrate the servers to the target environment. You need to make decisions about the target environment. You can get some of this metadata from discovery tools. The following are some examples of this metadata type:
  - Target subnet

  - Target security group
  - Target instance type
  - Target AWS Identity and Access Management (IAM) role
  - Target IP address
  - Target AWS account ID
  - Target AWS Region
  - Target AWS service
  - Target application architecture design
- **Wave planning metadata** – Wave planning metadata is the metadata type that helps you manage the migration. The following are examples of this metadata type:
  - Wave ID
  - Wave start time
  - Wave cutover time
  - Wave owner
  - Wave to application/server/database/move group mapping

Validate your discovery data

It is important to understand your current discovery data before making any decisions. You likely don't have all of the information at this stage of the migration. This playbook helps you define the metadata requirements and helps you collect the metadata efficiently. Ask yourself the following questions to identify what metadata is currently available and where it might be located:

- Have you used any tools to conduct a migration assessment, such as Migration Evaluator?
- Have you deployed any discovery tools in your environment, such as AWS Application Discovery Service or Flexera One Cloud Migration and Modernization?
- Do you have a CMDB that has the most up-to-date information for your IT portfolio?
- Have you finished the initial portfolio assessment in the mobilize phase?
- Have you completed the initial wave planning?
- Have you completed the initial target environment design?
- What is the source of each metadata type?
- Do you have access to all the metadata?
- How do you access all the metadata?
- Have you documented the process of accessing metadata?

Step 2: Identify the business and technical drivers

Business and technology drivers are critical when considering the high-level migration strategies and patterns for each application. You must understand the drivers that are unique to your migration. You use these business and technical drivers when validating your migration strategies and defining application mapping rules.

Common business drivers

Business drivers are factors related to business goals or limitations that you must consider when planning a large migration, such as contracts expiring, rapid growth, or budget. The following are common business drivers:

- **Exiting a data center** – You need to migrate as quickly as possible to the cloud. For example, a data center contract is about to expire.
- **Reducing operational cost and risks** – You want to reduce the costs or risks associated with operating an on-premises environment.
- **Flexibility** – You need to move to the cloud as a strategic direction in order to prepare for change in the business's future.
- **Growing the business** – You need to be able to quickly accelerate development and innovation or accommodate rapid growth.
- **Using data intelligently** – You want to take advantage of cloud-based artificial intelligence, machine learning, and Internet of Things (IoT) that can forecast your company's growth and provide insights into customer behavior.
- **Improving security and compliance** – You need to leverage the compliance programs that are already built into the AWS Cloud infrastructure, or you want to leverage the software-based security tools that can warn you of a possible threat to your data.
- **Resource availability** – Having limited resources or limited internal experience might lead you to select strategies that move the application without modification.

Common technical drivers

Technical drivers are factors related to technical goals or limitations that you must consider when planning a large migration, such as the current architecture. The following are common technical drivers:

- **Hardware or software end-of-support** – Your hardware or software is close to the end of its lifecycle, and you need to refresh it because the vendor no longer supports it.
- **Technology integration** – You gain access to global infrastructure that enables you to rapidly and strategically scale your application. You can go global quickly with global services and infrastructure ready for you to tap.
- **Storage and compute limitations** – Your data center does not have capacity for more storage or servers, and you need to find another place to expand.
- **Scalability and resiliency requirements** – Your applications have experienced downtime in the past, and you want to use the cloud to improve the recovery point objective (RPO) and recovery time objective (RTO).
- **Modernizing application architecture** – You want to take advantage of the cloud and change your applications to be cloud-native.
- **Improving performance** – Your application performance is poor during peak seasons, you want to scale up and down automatically to match the demand.

Update the runbook

1. In the portfolio playbook templates, open the *Runbook template for application prioritization* (Microsoft Word format).
2. In the *Business and technical drivers* section, record the drivers you identified for your large migration project.
3. Save your application prioritization runbook.

Step 3: Validate the migration strategies

Selecting migration strategies is critical to a large migration. You must verify that the migration strategies you select meet organizational expectations, limitations, and requirements. For more information about the available migration strategies, see the Guide for AWS large migrations.

You might have selected migration strategies in the mobilize phase or during the initial portfolio assessment. In this step, you use the business and technical drivers in order to select and validate the migration strategies for your portfolio.

Your migration strategies might change as you continue to assess the portfolio and begin the migration. At this stage, the goal is to understand the general distribution of your portfolio to each migration strategy. Selecting migration strategies is critical to the next step, validating the detailed migration patterns.

Select and validate the migration strategies

Evaluate the portfolio and select the migration strategies as follows:

1. Review all of the technical and business drivers you identified in the previous step, and prioritize the drivers based on your business needs.
2. Map each business and technical driver to a migration strategy. The following table is an example.

Priority	Business or technical driver	Migration strategy
1	Exit a data center by a specified date	Rehost as many applications as possible, and replatform and refactor only if rehost is not possible.
2	Reduce operational cost and risks	To accelerate the migration, rehost as many applications as possible.
3	Hardware or software end-of-support	Rehost supported applications and replatform applications that are not supported to newer hardware and software in the cloud.
4	Resource availability	Relocate to VMware Cloud or rehost to AWS Managed Services (AMS) to reduce the operational overhead.

3. By weighing each business and technical driver and evaluating your portfolio at a high level, estimate how the applications should be distributed amongst each migration strategy. It is common to see conflicts between the drivers. Project stakeholders need to work together and make final decisions to resolve the conflicts. The following is an example of how you might distribute your portfolio to each migration strategy:
   - Rehost – 60%
   - Replatform – 15%
   - Retire – 10%
   - Retain – 5%
   - Repurchase – 5%
   - Refactor – 5%

Do not proceed with the migration until you have selected high-level migration strategies for your portfolio.

Update the runbook

1. Open your application prioritization runbook.
2. In the *Migration strategies* section, record how the application workload is distributed among the seven migration strategies. For example:
   - Rehost – 60%
   - Replatform – 15%
   - Retire – 10%
   - Retain – 5%
   - Repurchase – 5%
   - Refactor – 5%
3. Save your application prioritization runbook.

Step 4: Validate the migration patterns

About migration patterns

A *migration pattern* is a repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. An example is *Rehost to Amazon Elastic Compute Cloud (Amazon EC2) using AWS Application Migration Service*. The following AWS services and solutions are frequently referenced in common migration patterns:

- AWS App2Container
- AWS Application Migration Service (AWS MGN)
- AWS CloudFormation
- AWS Database Migration Service (AWS DMS)
- AWS DataSync
- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic File System (Amazon EFS)
- AWS CloudEndure Migration Factory Solution (CEMF)
- Amazon Relational Database Service (Amazon RDS)
- AWS Schema Conversion Tool (AWS SCT)
- AWS Transfer Family

Similar to selecting migration strategies, you might have already identified your migration patterns in an earlier phase. However, you must validate them and make sure the patterns have been defined and documented. The following table lists common migration strategies and patterns.

ID	Strategy	Pattern
1	Rehost	Rehost to Amazon EC2 using Application Migration Service or CEMF
2	Replatform	Replatform to Amazon RDS using AWS DMS and AWS SCT
3	Replatform	Replatform to Amazon EC2 using AWS CloudFormation

ID	Strategy	Pattern
		Note CloudFormation templates build new infrastructure in the AWS Cloud
4	Replatform	Replatform to Amazon EFS using AWS DataSync or AWS Transfer Family
5	Replatform	Replatform to Amazon ECS using AWS App2Container
6	Replatform	Replatform mainframe or midrange servers to Amazon EC2 using an emulator
7	Replatform	Replatform from Windows to Linux on Amazon EC2
8	Retire	Retire the application
9	Retain	Retain on premises
10	Repurchase	Repurchase and upgrade to SaaS
11	Relocate	Relocate to VMware Cloud on AWS using VMware HCX
12	Refactor or re-architect	Re-architect the application

Update the runbook

At this point, you define the patterns at the portfolio level. Later in this playbook, you map each application to its corresponding migration pattern.

1. Open your application prioritization runbook.
2. In the *Migration patterns* section, record the migration patterns you have identified and validated. Assign each pattern a unique ID and note the migration strategy for the pattern.
3. Save your application prioritization runbook.

Note that migration patterns might change as you progress. You might change your migration strategies and patterns later, as you find new information, change the scope of the workload, or even decide to use new AWS services.

Task exit criteria

If you have not yet identified your migration strategies and patterns from a high-level, portfolio perspective, we highly recommend you work with the technical teams to define them before moving on to the next task. Portfolio assessment and wave planning are dependent on understanding the migration strategies and patterns. You do not need to have a comprehensive list of migration patterns before you proceed. You can add new patterns and adjust your strategies as you go.

Continue to the next task when you have completed the following:

- You have access to the latest discovery data and understand it.
- You have identified the business and technical drivers for your migration.
- You have selected and validated migration strategies, based on your business and technical drivers.
- You have selected and validated migration patterns.
- You have documented the following in your application prioritization runbook:
  - Business and technical drivers
  - Migration strategies
  - Migration patterns

Task 2: Defining processes for identifying, collecting, and storing metadata

In the previous task, you validated the initial discovery data, the migration strategies, and the migration patterns for your large migration. In this task, you identify what metadata is required and decide how you will collect it. This task consists of the following steps:

As you complete the steps in this section, consider the entire migration cycle from a metadata perspective. Consider portfolio assessment, wave planning, migration, testing, post-cutover activities, and then analyze all possible use cases and related use cases. Thinking about the information that you need to complete the full migration process helps you identify all of the metadata for that pattern.

Step 1: Define the required metadata

Before you can determine the required metadata attributes, you must understand the migration pattern. For example, you need different metadata for migrating a server to Amazon EC2 and for migrating a database to Amazon RDS. Most patterns are made up of many small tasks. In order to perform the migration pattern, you need to know what metadata attributes are required and then collect the metadata for that application. You must determine and gather the required metadata in the initialization stage so that you can perform the migration efficiently and without delay in the implementation stage.

The person or team that defines the metadata attributes begins by defining the steps and tasks needed to perform the migration pattern. The tasks determine what metadata is needed, so working through each task builds a comprehensive collection of the required metadata. The person who determines what metadata is required typically needs to have a comprehensive understanding of how to complete the migration pattern. Coordination with the person writing the migration runbook might be required. For more information, see the Migration playbook for AWS large migrations.

During a large migration, there are many processes spread across all workstreams that have a dependency on metadata. Having timely and accurate metadata has broad and significant impact to the success of a large migration.

In this step, you define the pattern or task and then use the definition to identify the metadata required.

Identify the key components of the migration patterns and supporting tasks

In this step, for each migration pattern or supporting task, you define the key components, such as the action, source object, target object, and tools used. You then name the pattern or task based on your answers.

Supporting tasks include the operational activities that the portfolio and migration workstreams need to perform during the migration, such as wave planning, application prioritization, dependency analysis, governance, disaster recovery, performance testing, or user-acceptance testing. Because you need metadata to support these tasks, you perform these steps for both the migration patterns and the supporting tasks.

1. **Action** – Identify the migration strategy or supporting task. Remember that one action might have other actions associated with it. For example, you might want to define operations for migration. Example actions include:
   - Migration strategy, such as rehost, replatform, or relocate
   - Wave planning
   - Application prioritization and dependency analysis
   - Operation
   - Governance
   - Disaster recovery
   - Testing, such as performance testing or user-acceptance testing (UAT)
2. **Source object** – Identify the source object on which the action will be performed. Example source objects include:
   - Waves
   - Server
   - Database
   - File share
   - Application
3. **Tools** – Identify the services or tools used to perform the action. You might use more than one tool or service. Example tools include:
   - AWS Application Migration Service
   - AWS DataSync
   - AWS Database Migration Service (AWS DMS)
   - AWS Backup
   - Performance monitoring tools
4. **Target object** – Identify the target object, service, or location where the source will reside when the action is complete. Example objects, services, or locations include:
   - Amazon Elastic Compute Cloud (Amazon EC2)
   - Amazon Relational Database Service (Amazon RDS)
   - Amazon Elastic File System (Amazon EFS)
   - Amazon Elastic Container Service (Amazon ECS)
   - Wave plan
5. **Pattern name** – Combine your answers to the previous steps as follows:

   <action> <source object> on/to <target object> using <tool>

   The following are examples:

- Rehost (action) waves, applications, or servers (source object) to Amazon EC2 (target object) using Application Migration Service or CloudEndure Migration Factory (CEMF) (tools)
- Replatform (action) file shares (source object) to Amazon EFS (target object) using DataSync (tool)
- Replatform (action) databases (source object) to Amazon RDS (target object) using AWS DMS (tool)
- Performance monitoring (action) of applications (source object) on Amazon EC2 (target object) using Amazon CloudWatch (tool)
- Back up (action) servers (source object) on Amazon EC2 (target object) using AWS Backup (tools) after migration
- Wave planning (action) waves, applications, or servers (source object) to create a wave plan (target object)

The following is an example of how you might record *Pattern 1: Rehost to Amazon EC2 using Application Migration Service or CEMF* from the migration patterns table (p. 9).

Pattern ID	1
Pattern name	Rehost to Amazon EC2 using Application Migration Service or CEMF
Action	Rehost migration
Source object	Waves, applications, or servers
Tools	Application Migration Service or CEMF
Target object	Amazon EC2

Determine the metadata required for each pattern or task

Now that you have defined the pattern or task, you determine the metadata required for the source object, target object, tools, and other business information. To explain this process, this playbook uses *Pattern 1: Rehost to Amazon EC2 using Application Migration Service or CEMF* from the migration patterns table (p. 9) as an example. Note that for some patterns or tasks, some steps might not apply.

1. **Analyze the target object** – Working backwards from the target object, manually create the object and identify the metadata needed to support it. Capture the metadata as demonstrated in the following table.

   For example, when you create an EC2 instance, you must choose an instance type, storage type, storage size, subnet, security group, and tags. The following table includes examples of metadata attributes that you might need if your target object is an EC2 instance.

Attribute name	Object type	Description or purpose
`target_subnet`	Target EC2 instance	Subnet of the target EC2 instance
`target_subnet_test`	Target EC2 instance	Test subnet of the target EC2 instance
`target_security_group`	Target EC2 instance	Security group of the target EC2 instance

Attribute name	Object type	Description or purpose
`target_security_group_test`	Target EC2 instance	Test security group of the target EC2 instance
`IAM_role`	Target EC2 instance	AWS Identity and Access Management (IAM) role of the target EC2 instance
`instance_type`	Target EC2 instance	Instance type of the target EC2 instance
`AWS_account_ID`	Target EC2 instance	AWS account to host the target EC2 instance
`AWS_Region`	Target EC2 instance	AWS Region to host the target EC2 instance

2. **Analyze the tools** – Use the tool to create a target object and check for differences. Capture the tool-specific metadata as demonstrated in the following table, and remove the attributes from the previous table if it is not supported by the migration tool. For example, you cannot customize the OS type and storage size for Application Migration Service because the rehost migration tool is like-for-like. Therefore, you would remove target OS and target disk size if these attributes were included in the previous table. In the previous example table, all attributes are supported by the tool, so no action is required.

   The following table includes examples of metadata that you might need for the tools.

Attribute name	Object type	Description or purpose
`AWS_account_ID`	Tools (Application Migration Service)	AWS account ID for AWS Application Migration Service
`AWS_Region`	Tools (Application Migration Service)	AWS Region for Application Migration Service
`replication_server_subnet`	Tools (Application Migration Service)	Subnet for the Application Migration Service replication server
`replication_server_security_group`	Tools (Application Migration Service)	Security group for the Application Migration Service replication server

3. **Analyze the source object** – Determine the required metadata for the source object by assessing the actions as follows:

   - To migrate servers, you need to know the source server name and fully qualified domain name (FQDN) in order to connect to the server.
   - To migrate applications along with their servers, you need to know the application name, application environment, and application-to-server mapping.
   - To perform a portfolio assessment, prioritize applications, or define a move group, you need to know the application-to-server mapping, application-to-database mapping, and application-to-application dependencies.
   - To manage waves, you need to know the wave ID and the start and end times of the wave.

   The following table includes examples of metadata that you might need for the source object.

Attribute name	Object type	Description or purpose
`wave_ID`	Source wave	ID of the wave (for example: wave 10)
`wave_start_date`	Source wave	Start date for the wave
`wave_cutover_date`	Source wave	Cutover date for the wave
`wave_owner`	Source wave	Owner of the wave
`app_name`	Source application	Source application name
`app_to_server_mapping`	Source application	Application-to-server relationship
`app_to_DB_mapping`	Source application	Application-to-database relationship
`app_to_app_dependencies`	Source application	External dependencies of the application
`server_name`	Source server	Source server name
`server_FQDN`	Source server	Fully qualified domain name of the source server
`server_OS_family`	Source server	Operating system (OS) family of the source server (for example: Windows or Linux)
`server_OS_version`	Source server	OS version of the source server (for example: Windows Server 2003)
`server_environment`	Source server	Environment of the source server (for example: development, production, or test)
`server_tier`	Source server	Tier of the source server (for example: web, database, or application)
`CPU`	Source server	Number of CPUs in the source server
`RAM`	Source server	RAM size of the source server
`disk_size`	Source server	Disk size of the source server

4. **Consider other attributes** – In addition to the primary action, consider other actions and attributes related to the target object or application. For the example pattern, *Pattern 1: Rehost to Amazon EC2 using Application Migration Service or CEMF*, the action is rehost, and the target object is Amazon EC2. Other related actions for this target object might include backing up to Amazon EC2, monitoring the EC2 instance after the migration, and using tags to manage costs associated with the EC2 instance. You might also want to consider other application attributes that help you manage the migration, such as the application owner, who you might need to contact for questions or cutover purposes.

The following table includes examples of additional metadata that are commonly used. This table includes tags for your target EC2 instance. For more information about tags and how to use them, see Tag your Amazon EC2 resources in the Amazon EC2 documentation.

Attribute name	Object type	Description or purpose
`Name`	Target EC2 instance (tag)	Tag to define the name of a target EC2 instance
`app_owner`	Source application	The owner of a source application
`business_unit`	Target EC2 instance (tag)	Tag to identify the business unit for a target EC2 instance (for example: HR, finance, or IT)
`cost_center`	Target EC2 instance (tag)	Tag to identify the cost center for a target EC2 instance

5. **Create a table** – Combine all of the metadata identified in the previous steps into a single table.

Attribute name	Object type	Description or purpose
`wave_ID`	Source wave	ID of the wave (for example: wave 10)
`wave_start_date`	Source wave	Start date for the wave
`wave_cutover_date`	Source wave	Cutover date for the wave
`wave_owner`	Source wave	Owner of the wave
`app_name`	Source application	Source application name
`app_to_server_mapping`	Source application	Application-to-server relationship
`app_to_DB_mapping`	Source application	Application-to-database relationship
`app_to_app_dependencies`	Source application	External dependencies of the application
`AWS_account_ID`	Tools (Application Migration Service)	AWS account to host the target EC2 instance
`AWS_Region`	Tools (Application Migration Service)	AWS Region to host the target EC2 instance
`replication_server_subnet`	Tools (Application Migration Service)	Subnet for the Application Migration Service replication server
`replication_server_security_group`	Tools (Application Migration Service)	Security group for the Application Migration Service replication server
`server_name`	Source server	Source server name

Attribute name	Object type	Description or purpose
`server_FQDN`	Source server	Fully qualified domain name of the source server
`server_OS_family`	Source server	Operating system (OS) family of the source server (for example: Windows or Linux)
`server_OS_version`	Source server	OS version of the source server (for example: Windows Server 2003)
`server_environment`	Source server	Environment of the source server (for example: development, production, or test)
`server_tier`	Source server	Tier of the source server (for example: web, database, or application)
`CPU`	Source server	Number of CPUs in the source server
`RAM`	Source server	RAM size of the source server
`disk_size`	Source server	Disk size of the source server
`target_subnet`	Target server	Subnet of the target EC2 instance
`target_subnet_test`	Target server	Test subnet of the target EC2 instance
`target_security_group`	Target server	Security group of the target EC2 instance
`target_security_group_test`	Target server	Test security group of the target EC2 instance
`instance_type`	Target server	Instance type of the target EC2 instance
`IAM_role`	Target server	AWS Identity and Access Management (IAM) role of the target EC2 instance
`Name`	Target server (tag)	Tag to define the name of a target EC2 instance
`app_owner`	Source application	The owner of a source application
`business_unit`	Target server (tag)	Tag to identify the business unit for a target EC2 instance (for example: HR, finance, or IT)
`cost_center`	Target server (tag)	Tag to identify the cost center for a target EC2 instance

6. **Repeat** – Repeat this process until you have documented the required metadata for each pattern.

Step 2: Build the metadata storage and collection processes

In the previous step, you defined the metadata required to support your migration. In this step, you build a process for collecting and storing the metadata. This step consists of two tasks:

1. Analyze the required metadata from the previous step and identify the source.
2. Define a process for efficiently storing and collecting the metadata.

Analyze the metadata sources

There are many common metadata sources. Usually, the first thing you can access is a high-level asset inventory, which is typically exported from a configuration management database (CMDB) or from another existing tool. However, you need to collect metadata from other sources as well, using both automated and manual processes.

The following table contains common sources, the standard collection process for that source, and the common metadata types that you can expect to find from that source.

Metadata source	Collection type	Metadata type
Discovery tools	Automated	Source server
CMDB	Automated	Source server
Inventory from other tools, such as VMware vSphere	Automated	Source server
Application owner questionnaire	Manual	Source server, target server, wave
Application owner interview	Manual	Source server, target server, wave
Application design documentation	Manual	Target server
Landing zone design documentation	Manual	Target server, tools

After listing all the possible sources of your metadata, you analyze the metadata type and map each source to the metadata attributes that you identified in the previous step.

1. Get a complete list of metadata attributes from Step 1: Define the required metadata (p. 11).
2. Analyze each metadata type and determine which types cannot be retrieved using an automated process. This is usually the **target server metadata** and **wave metadata** types because these require decisions from the application owners. For example, which subnet and security group will you use for the target EC2 instances?
3. Analyze each metadata attribute and map it to a metadata source in the previous table. It is common to have a combination of multiple sources. You can use discovery tools to collect some source server

metadata. For information about using discovery tools to collect metadata, see Get started with automated portfolio discovery on the AWS Prescriptive Guidance website.

4. Create a table to map the metadata attribute to its type and source. The following table is an example.

Metadata attribute	Metadata type	Metadata sources
`app_name`	Source application	CMDB
`app_owner`	Source application	CMDB
`app_to_server_mapping`	Source application	CMDB, discovery tools, or application owner questionnaire
`app_to_DB_mapping`	Source application	CMDB, discovery tools, or application owner questionnaire
`app_to_app_dependencies`	Source application	CMDB, discovery tools, or application owner questionnaire
`server_name`	Source server	CMDB
`server_FQDN`	Source server	CMDB
`server_OS_family`	Source server	CMDB
`server_IP`	Source server	Discovery tools
`disk_size`	Source server	Discovery tools
`instance_type`	Target server	Discovery tools
`target_subnet`	Target server	Application owner questionnaire
`target_security_group`	Target server	Application owner questionnaire
`AWS_Region`	Target server	Application owner questionnaire
`AWS_account_ID`	Target server	Application owner questionnaire
`replication_server_subnet`	Tools (Application Migration Service)	Landing zone design documentation
`replication_server_security_group`	Tools (Application Migration Service)	Landing zone design documentation
`Name`	Target server (tag)	Application owner questionnaire
`business_unit`	Target server (tag)	Application owner questionnaire

Metadata attribute	Metadata type	Metadata sources
`cost_center`	Target server (tag)	Application owner questionnaire
`wave_ID`	Wave planning	Application owner interview
`wave_start_date`	Wave planning	Application owner interview
`wave_cutover_date`	Wave planning	Application owner interview

Define a single metadata store

After mapping each metadata attribute to its source, you define where to store the metadata. Regardless of how and where you store the metadata, you need to choose only one repository. This ensures that you have a single source of truth. Storing metadata in multiple places is a common mistake in large migrations.

Option 1: Store metadata in a spreadsheet in a shared repository

Although this option might sound like a very manual process, it is the most common data store for large migrations. It is also common to store the spreadsheet in a shared repository, such as a Microsoft SharePoint site.

A Microsoft Excel spreadsheet is easy to customize and doesn't take a long time to build. The disadvantages are that it will get very complex if you have a lot of metadata and that it can be difficult to manage the relationships between assets, such as between the server, application, and database. The other challenge is version management. You need to limit write access to only a few people, or you need to use an automated process to update the spreadsheet.

In the portfolio playbook templates, you can use the *Dashboard template for wave planning and migration* (Excel format) as a starting point for building your own data store spreadsheet.

Option 2: Store metadata in a purpose-built tool

You can use a prebuilt tool, such as TDS Transition Manager (TDS website), to store your data, or you can build your own tool. When you build your own tool, you need database tables just like Excel spreadsheet tabs in option 1. For example:

- Server table
- Application table
- Database table
- Application-to-server and application-to-database mapping table
- Wave-planning table
- Application owner questionnaire table

Define the metadata collection processes

In the previous steps, you mapped the metadata to its source and defined a data store where you will collect the metadata. In this step, you build processes to effectively collect the metadata. You should minimize the manual copy-and-paste process and use automation to collect the metadata from each source. There are three steps:

1. Build an extract, transform, and load (ETL) script for each metadata source based on the metadata mapping table.

2. Build a scheduled task that imports metadata from each source automatically on a regular basis.
3. Build an export process or provide application programming interface (API) access to the metadata stored in the repository.

The following table is an example of the metadata attributes collected by each ETL script. The metadata is stored in the location you defined in the previous section, such as a spreadsheet or purpose-built tool.

Metadata attribute	**Metadata type**	**Metadata source**	**Collection process**
`app_name`	Source application	CMDB	ETL script – CMDB
`app_owner`	Source application	CMDB	ETL script – CMDB
`app_to_server_mapping`	Source application	CMDB	ETL script – CMDB
`app_to_DB_mapping`	Source application	CMDB	ETL script – CMDB
`app_to_app_dependencies`	Source application	Discovery tool	ETL script – discovery tool
`server_name`	Source server	CMDB	ETL script – CMDB
`server_FQDN`	Source server	CMDB	ETL script – CMDB
`server_OS_family`	Source server	CMDB	ETL script – CMDB
`server_OS_version`	Source server	CMDB	ETL script – CMDB
`disk_size`	Source server	Discovery tool	ETL script – discovery tool
`instance_type`	Target server	Discovery tool	ETL script – discovery tool
`target_subnet`	Target server	Application owner questionnaire	ETL script – application owner
`target_security_group`	Target server	Application owner questionnaire	ETL script – application owner
`AWS_Region`	Target server	Application owner questionnaire	ETL script – application owner
`AWS_account_ID`	Target server	Application owner questionnaire	ETL script – application owner
`Name`	Target server (tag)	Application owner questionnaire	ETL script – application owner
`business_unit`	Target server (tag)	Application owner questionnaire	ETL script – application owner
`cost_center`	Target server (tag)	Application owner questionnaire	ETL script – application owner
`wave_ID`	Wave planning	Application owner questionnaire	ETL script – application owner
`wave_start_date`	Wave planning	Application owner questionnaire	ETL script – application owner

Metadata attribute	Metadata type	Metadata source	Collection process
`wave_cutover_date`	Wave planning	Application owner questionnaire	ETL script – application owner

Step 3: Document metadata requirements and collection processes in a runbook

In this task, you document your decisions in a *metadata management runbook*. During the migration, your portfolio workstream adheres to this runbook as the standard procedures for collecting and storing metadata.

1. In the portfolio playbook templates, open the *Runbook template for metadata management* (Microsoft Word format). This serves as a starting point for building your own runbook.
2. In the *Metadata attributes* section, create a metadata attributes table for each migration pattern, and populate the tables with the metadata attributes identified in Step 1: Define the required metadata (p. 11).
3. In the *Source locations* section, document the sources you identified in Analyze the metadata sources (p. 18).
4. In the *Source location access instructions* section, document the steps a user would need to follow in order to access the metadata source locations.
5. In the *Metadata store* section, document the steps a user would need to follow in order to access the metadata store you created in Define a single metadata store (p. 20).
6. In the *Data collection types* section, identify the data collection process that you will use for each metadata source. Ideally, you should automate all metadata collection by using automation scripts.
7. In the *Data collection by metadata attribute* section, for each metadata attribute, identify the following according to the instructions in Define the metadata collection processes (p. 20):
   a. Metadata type
   b. Metadata source
   c. Metadata store
   d. Collection type
8. In the *Collect metadata* section, update the process as needed for your use case. This is the process the portfolio workstream follows in the implementation stage when they collect metadata for waves.
9. Verify that your runbook is complete and accurate. This runbook should be a source of truth during the migration.

10Share your metadata management runbook with the team for review.

Task exit criteria

Continue to the next task when you have completed the following:

- You have prepared a single repository for storing the collected metadata.
- In your metadata management runbook, you have defined and documented the following:
  - The metadata attributes required for each migration pattern
  - Metadata sources and detailed instructions for how to access each source
  - The metadata store and detailed instructions for how to access it
  - The processes used to collect metadata

- A mapping table that maps metadata attributes to the metadata sources and collection processes

Task 3: Defining the application prioritization process

Application prioritization is the process of determining the order in which applications should be migrated to the cloud. You assess priority based on the complexity of migrating the application to the cloud and the rules that you define. When discussing application prioritization, high priority does not necessarily correlate with the importance of the application to the business. In fact, applications that are business-critical are typically low priority for the migration because business-critical applications have higher risks. In a large migration, you prioritize low-complexity applications that are not business critical, and with each wave, you migrate increasingly more complex or business-critical applications.

In a large migration, where you have hundreds of applications lined up for migration, we don't recommend that you prioritize and plan every application all at once. This is one of the reasons why defining an application prioritization process is critical to a large migration project. To approach the migration in an agile way, you can pick the highest priority applications (3–10 applications), or you can select enough applications for 3–5 waves. You then complete application discovery and wave planning for only the selected applications. This approach saves a significant amount of time because application priority and waves often change during the course of a large migration.

One common myth about application priority is that the highest priority applications should be in the first wave. When you are performing wave planning, there is a high possibility that only a few of the 10 highest priority applications will be in the first wave because the others are not ready. This could be for a variety of valid reasons, such as dependencies, business constraints, or resource availability. Application priority is a critical factor in wave planning, but it shouldn't be the only factor you consider.

In this task, you define the application prioritization process and rules. This task consists of the following steps:

- Step 1: Define the application prioritization process (p. 28)
- Step 2: Define the application prioritization rules (p. 30)
- Step 3: Finalize the application prioritization process (p. 31)

The next section discusses complexity scoring. This playbook provides three process options for how to prioritize applications, and two of the three options use complexity scoring. For more information about the process options, see Step 1: Define the application prioritization process (p. 28). If you plan on using the application nomination process, then you do not need to define complexity scoring criteria, and you should proceed directly to Step 1: Define the application prioritization process (p. 28).

About complexity scoring criteria

Complexity scoring is the process used to assess the difficulty of migrating an application, which is a critical factor when prioritizing applications. Complexity scoring involves evaluating all applications against the same set of business and technical criteria, which you define. When evaluating an application, you assign a score to each criterion. When you sum the scores of the business criteria and technical criteria, you get a complexity score that reflects the overall complexity of migrating that application. You can then use the complexity score when prioritizing applications and planning waves.

There are two categories of complexity scoring criteria:

- **Business criteria** – The criteria in this category relate to the business complexity of migrating the application, such as risk if the application becomes unavailable, security and compliance considerations, and resource availability.

- **Technical criteria** – The criteria in this category relate to the technical complexity of migrating the application, such as operating systems and their versions, the number of servers and users, and the migration strategy.

You should determine the appropriate scoring criteria for your use case. If you are manually scoring application complexity, in the portfolio playbook templates, the *Score sheet template for application complexity* (Microsoft Excel format) contains a standard set of criteria and score values. You might want to start with these values and then customize them for your use case. If you are using a discovery tool for application prioritization, these tools commonly include a standard set of criteria, and you can add, remove, or modify the criteria, and you can weight them according to your needs. When you are establishing criteria, use the questions in the next two sections to help refine your criteria.

Business criteria

The following are business criteria that are commonly used in complexity scoring.

Business criteria	Description
Business impact	Assess the impact to the business if this application becomes unavailable: • Does it have financial impact? • Does it have operations impact? • Does it impact the customer experience? • Does it impact a product or company event?
Staff availability	During the migration, you might need assistance from the application owner, a subject matter expert (SME), network or infrastructure administrators, testers, and developers. Assess the availability of these resources to help you during the migration: • Will this staff be available during the migration to assist and provide guidance to the migration teams? • Will this staff be available to test and validate the application after it's migrated? • Will this staff be available to provision IP addresses or ports necessary for running the application in the target environment?
Business complexity	Having many interdependent and interconnected stakeholders, information technology systems, and organizational structures can increase the business complexity. Assess business complexity as follows: • How long will it take for the business to approve the infrastructure and network changes, such as firewall changes or provisioning a new instance? • How long will it take for the business to approve new software or tool installations on their server, such as discovery tools?

Business criteria	**Description**
Readiness	Assess whether the application is ready for migration as follows: • Is the application currently undergoing or scheduled to undergo a technology refresh? • Is maintenance scheduled, and will it overlap with the scheduled migration? • Is the application scheduled for decommission? • Is the application currently being upgraded, and are any new features being developed or integrated?
Security	Assess the complexity of the application's security requirements and security policy as follows: • Do you need to provision IP addresses and ports for accessing the application? • Does the application require infrastructure protection? • Does the application require data protection? • Is key management required? • Does the application require special access management policies? • Does the application require monitoring or logging? • Does the application require an incident response process and automation? • Are alerts and notifications required for this application?

Business criteria	Description
Compliance	Compliance requirements might apply for the application, such as laws, regulations, and guidelines provided by the state, business industry, or company policy. Assess the complexity of the application's compliance requirements as follows: • Are there any data privacy and residency requirements? • Should data at rest in the application be encrypted? • Should data in transit to or from the application be encrypted? • Is audit logging required? • Does the application need to be compliant with accounting and financial standards, such as System and Organization Controls (SOC)? • Does the application need to be compliant with payment security standards, such as Payment Card Industry (PCI)? • Does the application need to be compliant with patient health information regulations, such as Health Insurance Portability and Accountability Act (HIPAA)? • Does the application need to be compliant with public-cloud security programs, such as Information System Security Management and Assessment Program (ISMAP)?
Application knowledge	Does someone in the organization, such as the application owner, have the knowledge, skills, and experience to maintain, integrate, troubleshoot, and fix issues? And are you able to extend the application to meet business demand?
Migration skills	Does staff in your organization have the skills to migrate the workload to the target environment?

Technical criteria

The following are technical criteria that are commonly used in complexity scoring.

Technical criteria	Description
Storage	Assess the application's current storage as follows: • Where is the application currently stored? Examples include a network-attached storage (NAS), a storage area network (SAN), or a local drive. • What is the total size of the current storage?

Technical criteria	**Description**
Number of users	How many users does this application have? You can use actual logs or estimate.
Server count	How many servers are in the application stack?
Connectivity	Assess how this application is connected to others in your organization as follows: • How many other applications are dependent on this application? • What is the impact to other applications if this application is unavailable?
Application OS and version	Assess the operating system (OS) and version of the application's server as follows: • Is the server's OS version no longer supported? • Is the server running a Unix or Linux OS? • Is the server running a Windows Server OS? • Is the application on a mainframe or on midrange servers?
Application dependencies	Assess how this application is dependent upon other resources in your environment: • What resources is this application dependent upon? Resources can be other applications, components, OS-specific services (such as registries or web servers), or libraries. • What is the impact to this application if one or more of those resources is unavailable?
Data migration	Assess whether you need to migrate data or files for this application: • How complex is the data migration? • How complex is the file migration?
Migration strategy	Assess the complexity of the selected migration strategy. For more information about migration strategies, see the Guide for AWS large migrations.
COTS or custom	Assess whether the application is custom made or commercial off-the-shelf (COTS) as follows: • Do you have the latest version of the source code? • Is the application vendor-supported? • Is the application outsourced?

Step 1: Define the application prioritization process

This playbook includes three process options for prioritizing applications. You can select one of the options, or you might decide to combine two or more and build a custom process. Evaluate your use case and determine which of the following is the best fit for your environment:

- Option 1: Manual complexity scoring (p. 28) – This is a manual prioritization process that can be completed by an individual or in a workshop-style session. In this process, you use complexity scoring criteria to assess the difficulty of migrating each application, which is an important factor in prioritizing applications. This manual process is well-suited to large migrations because it provides a consistent, quantitative approach for prioritizing a large application portfolio. However, evaluating each application against a defined set of criteria can be a slower process when compared with the other two options.
- Option 2: Application nomination (p. 29) – This is a manual prioritization process that is typically completed a workshop-style session. In this process, the application owners nominate applications for migration. To be successful, this process requires that the application owners have comprehensive knowledge of their respective applications. This process is recommended if time is a factor and you need to quickly prioritize applications.
- Option 3: Discovery tool (p. 30) – This is an automated prioritization process. If the discovery tool in your environment has an automated feature for automated application complexity scoring or prioritization, using this feature can save time and accelerate the application prioritization process. In this process, you typically define criteria within the parameters of the discovery tool, and then the tool analyzes the applications and provides a final complexity score. Before selecting this option, explore the features available in your discovery tool and verify that you can customize it to meet the needs of your use case.

Option 1: Manual complexity scoring

In this manual application prioritization process, you use a worksheet to evaluate the application against a defined set of complexity scoring criteria. We recommend that you complete the worksheet in a workshop-style session, or an individual can complete the worksheet by collaborating with stakeholders. You then use the final complexity score and the application prioritization rules in order to determine the application's priority. Of the manual processes, this provides the most consistent, quantitative approach to determining application complexity and using that information to prioritize applications.

For the complexity scoring steps in this process, we recommend you use the *Score sheet template for application complexity* (Excel format), available in the portfolio playbook templates. This template includes predefined business and technical criteria. You can add, remove, or modify these criteria, or you can adjust the scoring values. For example, you might prefer a scoring range of 1–10 instead of 1–5. Note the following about the provided template:

- You can hover over each criterion for a description of it.
- When you are familiar with the template, you should delete the examples. These are for demonstration purposes only.

Keep the complexity score sheet updated throughout the initialization and implementation stages of the migration. You might change scores as you progress through the portfolio assessment. The application deep dive is a common time to update the score sheet because you learn more about each application as the team examines it in detail. During the migration, you might also change the application priority if you encounter issues, such as undiscovered dependencies and limitations that prevent you from migrating the application. By maintaining the score sheet throughout the migration, you can prioritize applications with greater accuracy.

Document your application prioritization process as follows:

1. In the portfolio playbook templates, open the *Score sheet template for application complexity*.
2. On the **Applications** sheet, add, modify, or remove criteria as appropriate for your migration. When modifying the criteria, do the following:
   - Review the guidance in the About complexity scoring criteria (p. 23) section of this playbook.
   - Consider the impact each criterion has to the duration, resources, and cost of the migration.
   - For a reliable complexity score, include criteria that represent varying levels of migration complexity in your organization.
3. On the **Scoring guide** sheet, update the default values and criteria as needed for your use case.
4. Save the score sheet.
5. Open your application prioritization runbook.
6. In the *Application complexity scoring criteria* section, update the section to reflect the location of your score sheet.
7. In the *Application prioritization process* section, do the following:
   a. Keep *Option 1: Manual complexity scoring* and delete the other options.
   b. Modify the process as needed for your use case.
   c. Remove any headings in this section that contain the word *Option*. Leaving these in the runbook might confuse users into thinking that the process is optional or that multiple options are available.
   d. Save your application prioritization runbook.

Option 2: Application nomination

This manual application prioritization process is the easiest and fastest approach to prioritizing applications. In this process, you ask the application owners to nominate applications that can be easily migrated to the cloud. You and the application owners can then quickly prioritize the applications because you already have in-depth knowledge of the nominated applications. We recommend that you work with stakeholders in a workshop-style session, but you can also collaborate via email, shared documentation, and other communication platforms.

During the nomination process, you enter the nominated applications into the *Score sheet template for application complexity* (Excel format), included in the portfolio playbook templates. You will not use all of the scoring and criteria features in this template, but we recommend using this sheet to record the nomination and prioritization decisions.

In some situations, the application nomination process is used to accelerate the prioritization, and the score sheet might not be needed. For example, if you are prioritizing only a handful of applications or if the application owners are very knowledgeable about their applications, the application owners and stakeholders can prioritize the applications based on their knowledge and experience. In this case, they can skip using the score sheet, and proceed directly to prioritization.

Document your application prioritization process as follows:

1. Open your application prioritization runbook.
2. Delete the *Application complexity scoring criteria* section. This process does not use application complexity scoring.
3. In the *Application prioritization process* section, do the following:
   a. Keep *Option 2: Application nomination* and delete the other options.
   b. Modify the process as needed for your use case.
   c. Remove any headings in this section that contain the word *Option*. Leaving these in the runbook might confuse users into thinking that the process is optional or that multiple options are available.
4. Save your application prioritization runbook.

Option 3: Discovery tool

If your discovery tool has a feature for complexity scoring or application prioritization, this automated process requires few resources and can accelerate the application prioritization process. You customize the criteria in the discovery tool for your use case, and then the discovery tool automatically analyzes the applications and provides a final complexity score. Because the tool already has all of the application metadata, you don't need to enter it.

For example, the Flexera One Cloud Migration and Modernization (formerly Flexera Foundation and CloudScape) discovery tool has a complexity scoring features called the *Optimization Scorecard*. This feature allows you to select the criteria that you want to include in scoring and weigh each criterion based on your preference. After data discovery is complete, the discovery tool analyzes the data based on the weighted criteria you provided and, using the tool's proprietary formula, produces the final complexity scores. For more information, see the Foundation and CloudScape User Guide (Flexera documentation) and Optimization Scorecard (Flexera documentation).

The disadvantage of this process is that it requires time (4–8 weeks) to set up the scanning appliance for an agentless discovery tool in your environment or to install agents to all in-scope workloads. Before you can use the scoring feature in your discovery tool, you must allow additional time (4–12 weeks) for the discovery tool to collect metadata by scanning the application workloads and performing application stack analysis. However, you might find that the extra time needed to configure the discovery tool might be recovered by reducing the amount of time and resources needed for metadata collection and application prioritization. For example, if the discovery tool's data is still current, the portfolio workstream can reuse the discovery tool and its data from the mobilize phase to identify pilot applications.

Note
If you are using the discovery tool process, you can still use the manual *Score sheet template for application complexity* in order to analyze the application against a different set of criteria. This additional information might help you refine your application prioritization.

Document your application prioritization process as follows:

1. If you have not already done so, set up the discovery tool in your environment. For more information, see Get started with automated portfolio discovery on the AWS Prescriptive Guidance website.
2. Customize the complexity scoring or application prioritization criteria in your discovery tool according to the instructions for your tool. For more information about selecting criteria, see About complexity scoring criteria (p. 23).
3. Open your application prioritization runbook.
4. In the *Application complexity scoring criteria* section, update the section to reflect that the scoring criteria is defined in the discovery tool. Example: Complexity scoring criteria is defined in <your discovery tool>.
5. In the *Application prioritization process* section, do the following:
   a. Keep *Option 3: Discovery tool* and delete the other options.
   b. Modify the process as needed for your use case. It is important that you include step-by-step instructions for how to generate a report with the complexity scores. If available, you can include a link to the user guide.
   c. Remove any headings in this section that contain the word *Option*. Leaving these in the runbook might confuse users into thinking that the process is optional or that multiple options are available.
6. Save your application prioritization runbook.

Step 2: Define the application prioritization rules

In this step, you define *application prioritization rules*, which help you determine the migration order of the applications. Although an application's complexity score is an important factor in prioritizing applications and planning waves, business and technology factors also must be considered. You create rules to assess the priority of each application and help you schedule the application in the appropriate wave. Common business and technology rules include:

- Specifying the order and schedule for migrating data centers
- Prioritizing business units
- Capturing deadlines for critical business applications

Define your application prioritization rules as follows:

1. Open your application prioritization runbook.
2. In the *Application prioritization rules* section, add the custom rules for your migration.
3. Save the application prioritization runbook.
4. Maintain the rules in the application prioritization runbook. Rules are subject to change as the migration progress, scope changes, or schedules shift.

The following is an example set of application prioritization rules.

Priority	**Rule**
1	Applications in the New York data center should always have higher priority than applications in the Texas data center.
2	Applications in the IT department should always have higher priority than applications in the Marketing department.
3	Applications with high complexity scores should have higher priority.
4	SAP applications need to be migrated before the end of the year.

Step 3: Finalize the application prioritization process

Now, you define how the portfolio workstream uses the rules and processes to prioritize applications. This is the process that the portfolio workstream references in the implementation stage of the migration.

Customize this process in the application prioritization runbook as follows:

1. Open your application prioritization runbook.
2. In the *Stage 2: Prioritize applications* section, modify the process as appropriate for your use case and environment.
3. Save the application prioritization runbook.

Task exit criteria

Continue to the next task when you have completed the following:

- You have selected an application prioritization process from the available options.
- You have documented the following in your application prioritization runbook:
  - Application complexity scoring criteria (if applicable)
  - Application prioritization process
  - Application prioritization rules
- You have updated the *Stage 2: Prioritize applications* section of your application runbook.

Task 4: Defining the application deep dive process

Now that you are finished establishing rules and processes for application prioritization, you perform an application deep dive that will help you refine the priority of each application. You perform the application deep dive on one application at a time, in order from the highest to lowest priority. For projects with multiple portfolio teams, each team can perform an application deep dive on a different application at the same time.

During the deep dive, you might encounter some unexpected issues, such as dependencies, that affect the complexity of migrating the application. When this happens, you should modify your complexity scoring criteria you defined in the previous step and update the score sheet accordingly in order to get more accurate complexity scores for future applications. You can then update the application priorities by using the new complexity scores.

This task consists of the following steps:

- Step 1: Define the application workshop process (p. 32)
- Step 2: Define the application mapping process (p. 35)
- Step 3: (Optional) Define the application target state (p. 40)
- Step 4: Finalize the application deep dive process (p. 44)

Step 1: Define the application workshop process

The workshop process is one of the most efficient approaches to an application deep dive. Using this process, you collaborate with the stakeholders, application owners, and the portfolio team to assess and analyze the application. The goal is to clearly understand the current state of the application, including its architecture, business purpose, dependencies, and environment. You then use this detailed information about the size and complexity of the application to design the application's target state.

Each workshop typically lasts 1–8 hours, although you might find that additional time is needed for high-complexity applications. You can also break the workshop into multiple meetings, depending on the availability of resources, your preference, and the size and complexity of the application.

Identify the expected outcomes

Before conducting an application workshop, you should set an agenda and define the expected outcomes of the workshop, or information that you need to collect in the workshop. This allows the workshop participants to prepare for the workshop, helps keep the meeting on target, and ensures that by the end of the workshop, you have all of the information necessary to prioritize, wave plan, and migrate the application.

We recommend that you define a standard set of expected outcomes and document these in your application prioritization runbook. When preparing a workshop, you use the standard expected outcomes and add new ones for the specific application.

Define the standard set of expected outcomes for application workshops as follows:

1. Open your application prioritization runbook.
2. In the *Application workshop expected outcomes* section, establish a standard set of expected outcomes for application workshops. When preparing a workshop, you can customize these for the specific needs of the application.
3. Save the application prioritization runbook.
4. Maintain the expected outcomes in the application prioritization runbook. As you conduct application workshops and continue with portfolio assessment and wave planning, you might identify new expected outcomes or refine your existing outcomes.

The following are examples of expected outcomes for the application workshop.

Priority	**Expected outcome of application workshop**
1	Clear understanding of the application's current architecture, including associated servers, dependencies, the environment, and the application tier.
2	The team has collected the metadata to support the design of the target architecture. The following metadata is required: • Target AWS account ID • Target AWS Region • Target subnet • Target security groups
3	The application owner questionnaire is complete, and all key questions are answered.
4	The team has collected all application documentation, such as the user guide, application architecture documentation, testing documentation, design documentation, and application programming interface (API) documentation.

Define the application workshop rules

Before conducting an application workshop, you should define the rules that will govern your workshop. Common rules include workshop length, tools that might be needed in the workshop, and any scheduling considerations or deadlines that need to be considered. This helps keep the meeting on target and ensures that decisions made in the workshop align with the migration schedule.

We recommend that you document the application workshop rules in your application prioritization runbook.

Define your application workshop rules as follows:

1. Open your application prioritization runbook.
2. In the *Application workshop rules* section, define the rules that govern your workshops.
3. Save the application prioritization runbook.

4. Maintain the rules in the application prioritization runbook. As you conduct application workshops and continue with portfolio assessment and wave planning, you might identify new rules or refine your existing ones.

The following are examples of rules for the application workshop.

Priority	**Workshop rule**
1	Workshops should be scheduled for a maximum of 2 hours per session on Tuesdays and Thursdays.
2	There is a scheduled freeze of the infrastructure December 1–January 15.
3	There is a hands-on workshop for migration tools.
4	The data center contract will expire on March 31. Workloads must be evacuated by March 31 to avoid penalties and costly contract extensions.
5	Biometric application and time and attendance application will be retained.

Define the application workshop process

It is important that you define a standard process for conducting application workshops. This ensures a consistent experience and sets expectations for the workshop participants, which can improve the efficiency of the workshop.

There are three stages to the application workshop process:

- **Prepare for the workshop** – Preparing for the workshop helps ensure that the session goes smoothly and that participants know what is expected. To prepare for the workshop, you establish an agenda and define the expected outcomes, you identify participants, tools, and the information needed in the workshop, and you schedule the workshop. Scheduling the workshop at least one week in advance gives the team time to block their calendar, prepare for the workshop, and collect any useful information.
- **Conduct the workshop** – When conducting the workshop, you limit the discussion to the items on the agenda and ensure that you meet the expected outcomes. Note topics that you think are helpful but are not included in your agenda. It can be helpful to record the workshop.
- **Finalize the workshop outcomes** – After the workshop, your team should have a clear understanding of the current state of the application and potential pain points, risks, challenges, and blockers that could affect prioritization and migration. Common tasks after the workshop include: reprioritizing the application, drafting the future state of the application, and updating the runbook with any expected outcomes, rules, or process changes that might be helpful in the next workshop.

The *Runbook template for application prioritization* includes a standard step-by-step process for preparing, conducting, and finalizing an application workshop. Define your application workshop process as follows:

1. Open your application prioritization runbook.
2. In the *Application workshop process* section, modify the standard process to meet the needs of your use case.
3. Save the application prioritization runbook.

4. Maintain the process in the application prioritization runbook. As you conduct application workshops, you might identify changes that you want to make to this process.

Step exit criteria

- You have defined the workshop agenda and the resources and artifacts required to support the workshop.
- You have defined the expected outcome of the workshop and identified the information you need to collect in the workshop.
- You have conducted a trial of the workshop process and have the information needed to support application mapping and design the target state.
- You have documented the following in your application prioritization runbook:
  - Application workshop expected outcomes
  - Application workshop rules
  - Application workshop process

Step 2: Define the application mapping process

Application mapping is the process of assigning each application to a migration pattern, which you identified and validated in Step 4: Validate the migration patterns (p. 9). In this step, you define the rules that you will use to evaluate the application. You then define the process that you will use to evaluate each application. Mapping each application to the migration pattern's use case helps you identify the migration tool, any skills necessary to complete the migration, and the complexity of the migration pattern.

You do not always migrate an application to a single pattern. For more information about when you might need multiple patterns for the same application, see Define the application mapping process (p. 37) later in this section.

Application mapping rules

The application mapping rules help you evaluate the application and identify the appropriate migration pattern. Each rule consists of any information you should use to evaluate the application and match criteria for the pattern.

In the portfolio playbook templates, the *Runbook template for application prioritization* includes a section for documenting your application mapping rules. Define your process as follows:

1. Open your application prioritization runbook.
2. In the *Application mapping rules* section, define your application mapping rules.
3. Save the application prioritization runbook.
4. Maintain the rules in the application prioritization runbook.

The following table provides examples of application mapping rules.

Note
The pattern IDs and names in this table correspond to the sample patterns in Step 4: Validate the migration patterns (p. 9). Use the pattern IDs and names that you defined in your application prioritization runbook.

Priority	Mapping rule
1	Using utilization data or monitoring tools, identify whether the application is a zombie or idle application. If the application is a zombie or idle application, choose *Pattern 8: Retire the application*, and then shut down the servers in the application stack.
2	Identify whether migrating this application to the cloud provides business value. Applications that are used only on premises and are not shared across branch or geographical locations, such as time and attendance applications, typically do not need to be migrated to the cloud. If migrating this application does not provide business value, choose *Pattern 9: Retain on premises*.
3	Identify whether the operating system (OS) of the application is supported by AWS, AWS migration services, a vendor, or your rehost migration tool, and then do the following: • If the OS is supported, choose *Pattern 1: Rehost to Amazon EC2 using Application Migration Service or CEMF*. • If the OS is not supported, choose *Pattern 3: Replatform to Amazon EC2 using AWS CloudFormation*.
4	Identify whether the application is hosted in an on-premises VMware virtualization environment that will be migrated to VMware Cloud (VMC) on AWS and whether the application is compatible with VMware Hybrid Cloud Extension (HCX). If these criteria are met, choose *Pattern 11, Relocate to VMware Cloud on AWS using VMware HCX*.
5	Identify whether the application has a software as a service (SaaS) version or equivalent, and then evaluate the benefits and costs of moving to this new platform. If these criteria are met, choose *Pattern 10: Repurchase and upgrade to SaaS*.
6	Identify whether the application's on-premises database(s) can be migrated to a homogeneous service in the cloud, such as migrating an on-premises Oracle database to Amazon RDS for Oracle or migrating an on-premises MySQL database to Amazon Aurora MySQL-Compatible Edition database. If these criteria are met, use *Pattern 2: Replatform to Amazon RDS using AWS DMS and AWS SCT*.
7	Identify whether the application uses Microsoft .NET Core (.NET 5 or .NET 6), Java, PHP, or another open-source programming language and whether the application is hosted in Microsoft

Priority	Mapping rule
	Windows Server. Evaluate whether the cost of replatforming is justifiable. If these criteria are met, choose *Pattern 7: Replatform from Windows to Linux on Amazon EC2*.
8	Identify the on-premises local and shared file storage that your application is dependent on, and then determine whether it must be included in the migration. If the local and shared file storage must be migrated, choose *Pattern 4: Replatform to Amazon EFS using AWS DataSync or AWS Transfer Family*.
9	Identify whether the application's servers are mainframe or midrange servers, such as IBM AS/400 or Apache Spark, and confirm that the applications are compatible with emulators. If these criteria are met, use *Pattern 6: Replatform mainframe or midrange servers to Amazon EC2 using an emulator*.
10	Identify whether this is a legacy, monolithic, or mainframe-based application that can no longer meet the demand of the business due to its limitations. For example, identify whether the application can scale, integrate with related applications, or is expensive and difficult to maintain. If the application meets any of these criteria, choose *Pattern 12: Re-architect the application*.

Define the application mapping process

When you are mapping applications to the migration patterns, it is helpful to request discovery data for the application from the discovery team. This data typically includes information such as a recommended migration pattern (sometimes called the *R pattern* or *R score*), utilization information, application dependencies, and other information that you can use in the assessment. As you explore this application in detail, you might decide to change the migration pattern that was previously identified.

When you have the data, you can compare the application to the business and technical criteria that you identified in Step 2: Identify the business and technical drivers (p. 6). You recorded your drivers in your application prioritization runbook. Evaluating the application against the criteria might lead you to select multiple migration patterns for the application, or it might lead you to eliminate patterns based on cost, schedule, or other limitations.

The following is an example of a business driver that requires you to use multiple migration patterns on a single application. You want to migrate an on-premises SQL Server database to Amazon DynamoDB, but because the contract for the data center is expiring, the business would like to migrate it sooner than the proposed schedule to replatform it. To address this business driver, you revise the migration plan for the application into a two-pattern approach. First, you rehost the application to the cloud in order to remove it from the data center. Later, after the application is in the cloud, you can replatform it according to the proposed schedule.

You should also consider whether the application is an n-tier application, which is an application composed of multiple tiers. An *application tier* is a group of physical servers that host horizontal layers of your application. N-tier applications are more complex because each tier might require a different strategy, and you might choose to migrate the application tiers in different waves. For example, if you have an application that is composed of presentation, business service, and database tiers, there is a possibility that you can map a different pattern for each tier.

You then evaluate the application against your application mapping rules, which you defined in your application prioritization runbook. For more information, see Application mapping rules (p. 35) earlier in this section.

After you map your application to one or more patterns, review and verify this decision with the application owner. The application owner should confirm the selected pattern, and they should help you plan and perform the migration. At this time, application owners might also provide insights based as their experience or share any issues they anticipate with the migration.

When you have mapped the application to one or more migration patterns and confirmed the patterns with the application owner, you record the application, pattern ID, pattern name, and relevant drivers in an application mapping table in your application prioritization runbook.

In the portfolio playbook templates, the *Runbook template for application prioritization* includes a standard step-by-step process for application mapping. Define your process as follows:

1. Open your application prioritization runbook.
2. In the *Application workshop process* section, modify the standard process to meet the needs of your use case.
3. Save the application prioritization runbook.
4. Maintain the process in the application prioritization runbook.

The following table is a sample application mapping table. The provided *Runbook template for application prioritization* includes an empty table in which you can record your results from the application mapping process.

Note
The pattern IDs and names in this table correspond to the sample patterns in Step 4: Validate the migration patterns (p. 9). Use the pattern IDs and names that you defined in your application prioritization runbook.

Application name	Pattern ID	Pattern name	Business and technical drivers addressed
Corporate website	1	Rehost to Amazon EC2 using Application Migration Service or CEMF	• Data center exit • Reduce operational costs
HR system	8	Retire the application	• Reduce operational costs
Time and attendance application	9	Retain on premises	• Reduce operational costs • Reduce risk and impact
PO system	3	Replatform to Amazon EC2 using AWS CloudFormation	• Technology integration • Storage and compute limitation

Application name	Pattern ID	Pattern name	Business and technical drivers addressed
			• Hardware and software end-of-life support • Improve security and compliance
CRM system	10	Repurchase and upgrade to SaaS	• Reduce operational costs • Technology integration • Hardware and software end-of-life support • Accelerate development, innovation, and growth
Fixed asset system	7	Replatform from Windows to Linux on Amazon EC2	• Reduce operational costs
ERP file storage	4	Replatform to Amazon EFS using AWS DataSync or AWS Transfer Family	• Storage and compute limitation
Accounts payable system	11	Relocate to VMware Cloud on AWS using VMware HCX	• Data center exit • Technology integration • Reduce operational costs
Ledger system	6	Rehost mainframe or midrange servers to Amazon EC2 using an emulator	• Data center exit • Technology integration • Improve security and compliance • Hardware and software end-of-life support • Storage and compute limitation • Modernizing application architecture

Application name	Pattern ID	Pattern name	Business and technical drivers addressed
General ledger	12	Re-architect the application	• Reduce operational costs • Technology integration • Improve security and compliance • Hardware and software end-of-life support • Storage and compute limitation • Modernizing application architecture • Scalability and resiliency • Accelerate development, innovation, and growth

About AWS Migration Hub strategy recommendations

In addition to the application mapping process described, you can use the *Strategy Recommendations* feature in AWS Migration Hub to get recommended strategies as a reference. This feature is designed to help automate portfolio analysis and recommend migration and modernization strategies for your applications.

Strategy Recommendations analyzes your on-premises applications in order to determine their runtime environments and process dependencies. You can choose to include source code and databases in the analysis. You prioritize your business objectives, such as migration speed, licensing costs, and reducing operational costs. Strategy Recommendations assesses the collected information against your prioritized objectives and recommends viable paths for migrating and modernizing your applications. You then review the recommendations with the business to confirm that the recommend strategy meets the business and technical criteria.

Step exit criteria

- You have documented the following in your application prioritization runbook:
  - Application mapping rules
  - Application mapping process
- You have validated the mapping rules and processes by using one or more proof-of-concept (POC) applications.

Step 3: (Optional) Define the application target state

In this step, you define the attributes and the process that you use to document the target state, or *to-be state*, for the application. The target state is how the application operates in the target cloud environment after the migration. The target environment varies based on your target platform or service and business requirements. The target environment could be the AWS Cloud, AWS Managed Services (AMS), or VMware Cloud (VMC) on AWS.

Defining the target state helps project managers, migration consultants, architects, application owners, and stakeholders collaborate effectively. By using this process, the team can identify and resolve issues in advance and more efficiently implement the target state environment.

For some applications, this step is optional. You can skip this step if the application you are migrating is standalone or low-complexity. Migration strategies that don't modify the application, such as rehost, might not require this step. However, for more complex migration strategies, such as replatform and re-architect, you should define the target state before starting the migration.

The workshop provides you with an in-depth understanding of the current state of the application, so it's a good idea to draft the target state after completing the workshop. In addition, mapping the application to its migration pattern provides additional insights and helps you identify whether defining the target state is necessary. For example, if you map the application to the pattern *Rehost to Amazon EC2 using Application Migration Service or CEMF*, you've identified that the strategy is rehost, and you likely do not need to define the target state for this application.

Target state attributes

When defining the target state and making decisions about the application, we recommend that you consider the following target state attributes:

- **AWS Well-Architected Tool** – Review the application target state against the AWS Well-Architected Framework to help improve the security, performance, and resiliency of the application in the cloud.
- **Target landing zone** – Typically, by the end of the mobilize phase, you should have built a complete landing zone that is ready to run pilot applications. The landing zone should already be configured with a multi-account architecture, identity and access management, governance, data security, network design, and logging. You use a pilot application to verify that the landing zone is complete. Verify that you can launch and run your pilot application in the existing target landing zone. If you need to modify the landing zone for the application, inform the landing zone team of your requirements. For example, your application might require access to a service that is hosted in a separate account, or your application might require special routing to a virtual private cloud (VPC).
- **Dependencies** – Identify any applications that your application relies on in order to function properly. For example, your application might be dependent on databases, storage, or third-party services, such as a payment gateway or external web service, or your application might be dependent on other applications within your environment. In order to access these dependencies, you might need special routing or configuration, such as connecting to a directory service for authentication.
- **Dependent applications** – Identify any applications that rely on your application in order to function properly. You might need to reconfigure and update these applications in order to prevent downtime during the migration.
- **Security and compliance** – Review the target environment with the security and compliance team, and identify any gaps. The application might be composed of several components, logical layers, or multiple tiers. Depending on your compliance requirements, you might not be able to migrate some of these components to your target environment, or you might need additional security measures when migrating the workload. Common security and compliance requirements are data residency, encryption in transit, and encryption at rest. These require additional configuration in your target environment. For example, you might need to configure certificates in order to secure communications, or you might need encryption keys to secure the data at rest. You might also need to select multiple migration patterns for the application so that some application tiers remain on premises and other tiers are migrated to the cloud.
- **Storage dependencies** – Review your application storage dependencies and determine how migrating the application to the target environment would affect these dependencies. For example, if the application is dependent on network storage, such as network-attached storage (NAS) or a storage

area network (SAN), you need to plan for a storage service in the cloud, such as Amazon Simple Storage Service (Amazon S3) or Amazon FSx. You also need to plan how you will migrate your data to the target cloud environment in order to keep your application running.

- **Database** – Review the migration strategy for any databases the application uses. Are you going to replatform to a new database service, such as Amazon Relational Database Service (Amazon RDS), Amazon Aurora, or Amazon DynamoDB? Are you going to rehost your database in the target environment? In some cases, especially for a monolithic database, you need to refactor the database in order to address technical requirements, such as sub-second latency, or to take advantage of the features of a particular type of AWS database. As with data residency compliance requirements, you need to identify which data should be retained on premises and which should be migrated to the cloud. For example, you might need to retain an on-premises database table for customer information, and you could migrate the rest of the database to the cloud.
- **Application components** – Review the components your application depends on. Determine whether your application depends on a component that isn't supported by the target environment. If the target environment doesn't support all application components, you need to refactor the application to mitigate the issue. For example, if you have a .NET Framework application that is dependent on Windows-only components such as Component Object Model (COM) Interop, COM+, or the Windows registry, in order to replatform the application on a Linux operating system, you must refactor the application to .NET Core.
- **Application tiers** – Identify the number of tiers in your application. Is the application n-tier, two-tier, or standalone? Confirm that you understand the migration pattern for each tier. For example, your application might have a presentation (or *web*) tier that hosts the user interface, an application tier that hosts business services, and a database tier that hosts databases, and each tier might require a different migration pattern.
- **Disaster recovery** – Identify the application's current and future-state disaster recovery (DR) plan, including the recovery point objective (RPO) and recovery time objective (RTO). Decide whether to use the existing on-premises DR plan or to explore a new DR strategy in the cloud. For more information, see the sections Disaster recovery options in the cloud and Recovery objectives (RTO and RPO) in the *Disaster Recovery of Workloads on AWS: Recovery in the Cloud* whitepaper.

Define the target state process

To define the application target state, we recommend you use the provided template, *Application target state worksheet* (Excel format), which is available in the portfolio playbook templates. The template includes standard attributes that you can use or modify. Define the process for documenting the application target state as follows:

1. Open the *Application target state worksheet*.
2. Review the default attributes and make any changes for your use case.
3. Save the worksheet.
4. Open your application prioritization runbook.
5. In the *Target application state* section, do the following:
   a. In the *When to complete this process* section, establish criteria that allow the portfolio team to determine whether they need to define the application's target state.
   b. Update the attribute section as needed.
   c. Update the process section as needed for your use case.
6. Save the application prioritization runbook.

Samples of the application target state

The following table shows examples of how you use the *Application target state* worksheet to document the target state of the application.

Application	**Example** 1	**Example 2**
Target platform	AWS Cloud	VMC on AWS
Landing zone	Requires access to an on-premises directory service Requires AWS Control Tower to centralize management of multiple accounts and services across the organization	Requires network integration with an on-premises VMware vSphere environment **Note** We need to increase our bandwidth to support this new connection
Dependencies	Active Directory, payment gateway, inventory system	Ledger system, accounts payable, fixed asset system
Dependent applications	None	ERP, HR system
Security	Encryption at rest and in transit	Encryption at rest and in transit, allow only ERP and HR system
Compliance	PCI, SOC	SOC, FINMA
Storage dependencies	Boot drive attached, NAS, network share	Boot drive and data drive, NAS, SAN
Database	Current: Oracle DB Cloud: Amazon RDS for Oracle	Current: MySQL Cloud: Amazon Aurora MySQL-Compatible Edition
Application component	.NET Framework 4.5	Oracle Java SE 10
Application tiers	N-tier Front-end, business services, data services and agents, database	2-tier web application and database
Disaster recovery	RPO: 1 min, RTO: 5 min Current DR strategy is warm standby DR in any US regions	No downtime Current DR strategy is multi-site active/active Use existing, on-premises DR

Step exit criteria

- In the *Application target state worksheet*, you have defined the attributes that you want to include in the target state process.
- In your application prioritization runbook, you have done the following:
  - You have established criteria for when the portfolio team is expected to define the target state of the application.
  - You have updated the process for defining the target state based on your use case.

Step 4: Finalize the application deep dive process

Now, you define how the portfolio workstream uses the workshop, rules, and processes you established in this task to perform a deep dive into the application. This is the process that the portfolio workstream references in the implementation stage of the migration.

Customize this process in the application prioritization runbook as follows:

1. Open your application prioritization runbook.
2. In the *Stage 2: Perform application deep dive* section, modify the process as appropriate for your use case and environment.
3. Save the application prioritization runbook.
4. Share your application prioritization runbook with the team for review.

Task 5: Defining the wave planning process

Wave planning is a key milestone in a large migration. In a wave plan, you group similar applications together, accounting for infrastructure and application dependencies (such as a shared database), the priority of the applications, similarity of application architecture, and business functionality. You then review the wave plan with the application and infrastructure teams to confirm their availability during the specified migration and cutover window.

Based on real-life deployments for various AWS customers, the following are some of the best practices for wave planning:

- Plan migration waves at least 4–5 waves in advance. This helps ensure that there are always enough servers for the migration workstream.
- Fail fast. You should start with a few, low-complexity applications and apply your learnings to later waves.
- In early waves (waves 1–5), select fewer servers (less than 10), low-complexity applications, and applications in lower environments, such as development or test environments. Gradually introduce more complexity and more servers into the waves as you progress.
- Wave planning is an ongoing process, not a one-off task. Do not try to plan all waves at once.
- If you are using a portfolio discovery tool and it has a complexity scoring feature, use it in wave planning. Migrate the applications with the lowest complexity first.

This task consists of the following steps:

- Step 1: Define the move group process (p. 44)
- Step 2: Define the wave planning selection criteria (p. 46)
- Step 3: Finalize the wave planning process (p. 48)

Step 1: Define the move group process

In this step, you identify any application-to-server dependencies and define the rules that will be used to determine which servers should be moved together, as a move group. A *move group* is a block of servers or applications that should be moved together in a group. This is the building block of a migration wave, where each wave consists of one or more move groups, depending on the number of servers in each move group.

Identify the application dependencies

The following are key considerations when grouping interdependent applications in a move group:

- Consider infrastructure dependencies, such as:
  - An application might have multiple databases, and those databases could be shared by other applications.
  - An application might be dependent on another application.
  - A server might host databases for multiple applications.
- Consider business and operational dependencies, such as:
  - Due to a business impact or operational schedule (such as backup or patching), an application can only be migrated during a certain window.
  - An application owner is available only for one migration cutover window, so all of the owner's applications must be in the same move group.

You identified infrastructure dependencies in the application workshop process or when you defined the target state. You can identify infrastructure dependencies through automated or manual processes. To automate identification of infrastructure dependencies, you can use a discovery tool, such as Flexera One Cloud Migration and Modernization or TDS TransitionManager. For a manual process, validate CMDB information with the application and infrastructure teams.

You identified business and operational dependencies in the application workshop process.

As a starting point for building your own wave planning runbook, we recommend that you use the *Runbook template for wave planning* (Microsoft Word format) included in the portfolio playbook templates. Document the dependencies for your migration as follows:

1. Open your wave planning runbook.
2. In the *Application dependencies* section, record the dependency. Identify the type (infrastructure, business, or operational), the dependency, and a brief description of the dependency.
3. Save the wave planning runbook.
4. Maintain the dependencies in the wave planning runbook. As you progress, you might identify new dependencies.

The following table shows example dependencies.

Type	Dependency	Description
Infrastructure	Database	A database is shared with other applications
Infrastructure	File store	The application uses a central file store that can be decoupled, or all associated applications should migrate together
Infrastructure	Application	The application depends on one or more other applications to function, such as extract, transform, and load (ETL) jobs
Business	Business outage	Specific and approved outage windows for the application

Type	Dependency	Description
Operational	Patch window	Scheduled operational tasks, such as patching, that can impact migration cutover

Define the move group rules

After you record the dependencies in your wave planning runbook, you must build move group rules based on those dependencies. These rules govern how servers are grouped together into move groups. Use the following steps to build your rules:

1. Review the dependencies you defined in the previous section.
2. Choose the dependencies that affect whether applications must be moved together, in a move group. Not all dependencies require applications to be migrated together. For example, an infrastructure dependency on Microsoft Active Directory should not be considered when defining move groups because it is a common dependency for all applications. You should build a domain controller in the cloud before migrating any applications.
3. Convert the dependencies that require the applications to be moved together to a move group rule.

If an application matches any of the rules, then all of the associated servers must be placed in the same move group so that they are migrated together.

Document the move group rules for your migration as follows:

1. Open your wave planning runbook.
2. In the *Move group rules* section, record the move group rules in order of priority.
3. Save the wave planning runbook.
4. Maintain the rules in the wave planning runbook. As you progress, you might identify new rules.

The following table shows example move group rules.

Rule	Move group rule
1	Applications with a shared database must migrate together.
2	Applications that have the same application owner must migrate together.
3	Applications with the same patch window must migrate together.

Step 2: Define the wave planning selection criteria

After you have established move groups, you need to gather similar move groups together in order to form migration waves. In this step, you define the criteria that you use to select one or more move groups for each wave.

Understanding the size of each move group is critical to successful wave planning. The goal is to size each wave so that the migration remains agile and maintains a healthy pipeline of servers. Waves that are too large can be difficult to adapt to change in the migration plan, and waves that are too small might not provide sufficient servers to achieve the desired migration velocity.

We recommend that you consider the following criteria when sizing waves:

- **Small first waves** – Initial waves should be smaller, with less than 10 servers, and then you can gradually increase the number of servers in each wave. This allows you to fail fast and build on lessons learned. For example, migrate an application with 3 servers before migrating an application with 20 servers.
- **Resources** – Identify how many servers the migration team can migrate in a single wave. A standard measure is that a migration team of four architects can migrate up to 50 servers in a week for rehost patterns. Combine move groups to form migration waves that don't exceed the capacity of the migration team.
- **Agility** – Waves must be adaptive to any changes in the migration plan. In the event that you must reschedule a server, you should be able to reschedule the entire move group of affected servers.
- **Storage size** – Migrate smaller applications first. For example, migrate a 100 GB application before a 2 TB application.
- **Application environment** – Migrate applications in lower environments, such as development or test environments, before applications in production environments.
- **Application complexity** – Migrate less-complex applications with fewer external dependencies first.
- **Criticality of the application** – Migrate non-critical applications before mission-critical applications.
- **User base** – Migrate applications that have small user bases first. For example, migrate an application that has 10 users before an application that has 10,000 users.
- **Network bandwidth** – The size of the wave should not exceed the network bandwidth. For more information, see your migration principles, which you defined according to the instructions in the Foundation playbook for AWS large migrations.

Document the selection criteria for wave planning as follows:

1. Open your wave planning runbook.
2. In the *Wave planning selection criteria* section, record the criteria you want to use for your migration.
3. Save the wave planning runbook.
4. Maintain the criteria in the wave planning runbook. As you progress, you might need to adjust the criteria or add new criteria.

The following table shows example wave planning selection criteria.

Criteria	**Description**
Identify the least complex applications	Identify applications with higher complexity scores in the move groups.
Lower environment first	Non-critical applications within lower environments, such as development or test environments, must migrate first. Critical applications within production environments, such as those that generate revenue, must migrate last.
Fail fast	Form initial waves with less than 10 servers.
Migration team strength	Identify how many servers each migration team can cut over.

Criteria	Description
Combine similar move groups	Combine move groups based on commonalities. For example, the move groups might share the same application owner, source data center, or target AWS account.
Wave size	Waves should not exceed 50 servers in total.

Step exit criteria

- You have identified the wave planning criteria for your use case and documented it in your wave planning runbook.

Step 3: Finalize the wave planning process

Now that you have defined how to create move groups and established the criteria you used to combine move groups into migration waves, you must define the process for planning waves. In this step, you update your wave planning runbook to record the complete wave planning process, and you confirm that you have a dashboard tool the team can use to record wave information.

In this step, we recommend that you use the provided *Dashboard template for wave planning and migration*, which is available in the portfolio playbook templates. This template is designed to assist portfolio teams and serves as a starting point for collating data, helping to analyze application portfolios, identifying application-to-server dependencies, and eventually planning migration waves. You can modify this template as needed for your environment.

Document the wave planning process as follows:

1. Open the *Dashboard template for wave planning and migration*.
2. Modify the dashboard as needed for your use case. For example, you might add a worksheet for extracting server inventory, add a new pivot table or chart, or import source information by using the `VLOOKUP` function.
3. Save the dashboard template.
4. Open your wave planning runbook.
5. In the *Stage 2: Perform wave planning* section, modify the standard process provided to meet the needs of your use case.
6. Save the wave planning runbook.
7. Share your wave planning runbook with the team for review.
8. Maintain the process in the wave planning runbook. This process acts as a standard operating procedure for planning waves for your large migration.

Task exit criteria

- You have documented the following in your wave planning runbook:
  - Application dependencies
  - Application move group rules, listed in order of priority
  - Wave planning selection criteria
  - Wave planning process

Stage 2: Implementing a large migration

In stage 1, *initializing a large migration*, you defined the portfolio assessment and wave planning processes and documented them in runbooks. In stage 2, *implementing a large migration*, you complete these processes and repeat them for each sprint until the migration is complete.

The portfolio team completes the following portfolio assessment and wave planning tasks in stage 2:

- Task 1: Prioritizing the applications (p. 49)
- Task 2: Performing the application deep dive (p. 50)
- Task 3: Performing wave planning and metadata collection (p. 51)

Note
Portfolio assessment and wave planning is not a one-off task. It is a continuous task that supports the migration. You repeat all the tasks in this stage many times until the migration is complete.

The portfolio assessment and wave planning process typically requires 1–2 weeks for each wave. The portfolio workstream typically plans 4–5 waves in advance to maintain a healthy pipeline of servers for the migration workstream. The portfolio workstream begins planning waves at the end of the initialization stage (stage 1), and the implementation stage (stage 2) begins when the migration workstream begins migrating the first wave of applications. For an example of a wave schedule, see Stage 2: Implement a large migration in the *Guide for AWS large migrations*.

Tracking progress

As you begin preparing waves for migration, we recommend that you track the status of each application through the portfolio assessment process. In the portfolio playbook templates, you can use *Progress tracking template for portfolio assessment* (Microsoft Excel format). This template allows you to track the following for each application: complexity score, target wave, application owner, target completion dates for the primary tasks (application prioritization, deep dive, wave planning, and data collection), and the overall readiness of the application for migration. The guidance in this playbook includes instructions for when to update the progress tracking sheet.

Task 1: Prioritizing the applications

In this task, you review the list of unmigrated applications in your portfolio and, for a subset of the remaining applications, assign an application complexity score and priority. You repeat this process many times throughout the migration project.

You need the following information to complete this task.

Input	Source
A complete list of the applications in your portfolio that you are going to migrate	A discovery tool or a configuration management database (CMDB)

Input	Source
The target migration strategy and pattern, at a high level	*Migration strategies* and *Migration patterns* in your application prioritization runbook
The number of applications that you plan to include in the wave	*Wave planning selection criteria* in your wave planning runbook

Follow the instructions in your application prioritization runbook, in the *Stage 2: Prioritize applications* section. You defined this process in this playbook, in Step 3: Finalize the application prioritization process (p. 31).

At the end of this task, you have completed the following.

Output	Description
A list of prioritized applications	You have prioritized 2–3 times the number of applications that you plan to include in the wave, and you have entered these applications in the progress tracker.

Task 2: Performing the application deep dive

In this task, you perform a deep dive into each application you prioritized in the previous task. This usually includes sending a questionnaire to the application owner, analyzing any application dependencies, and scheduling an application workshop.

You need the following information to complete this task.

Input	Source
A list of prioritized applications	Created earlier in the implementation stage, in Task 1: Prioritizing the applications (p. 49)
The target migration strategy and pattern, at a high level	*Migration strategies* and *Migration patterns* in your application prioritization runbook

Follow the instructions in your application prioritization runbook, in the *Stage 2: Perform application deep dive* section. You defined this process in this playbook, in Step 4: Finalize the application deep dive process (p. 44).

At the end of this task, you have completed the following.

Output	Description
Migration pattern mapping	You have mapped each application to a migration pattern.
Application target state (if applicable)	If applicable for the application, you have defined the future state of the application in the cloud.

Task 3: Performing wave planning and metadata collection

This is the final task for portfolio assessment and wave planning. In this task, you use the application information and target migration pattern to build move groups, assign move groups to waves, and collect all the metadata needed to support the migration. Finally, you notify the migration workstream that the wave is ready.

You need the following information to complete this task.

Input	Source
A list of prioritized applications	Created earlier in the implementation stage, in Task 1: Prioritizing the applications (p. 49)
Migration pattern mapping	Created earlier in the implementation stage, in Task 2: Performing the application deep dive (p. 50)
Application target state (if applicable)	Also created in Task 2: Performing the application deep dive (p. 50)

Do the following:

1. Follow the instructions in your wave planning runbook, in the *Stage 2: Perform wave planning* section. You defined this process in this playbook, in Step 3: Finalize the wave planning process (p. 48).
2. Follow the instructions in your metadata management runbook, in the *Stage 2: Collect metadata* section. You defined this process in this playbook, in Step 3: Document metadata requirements and collection processes in a runbook (p. 22).
3. Notify the migration workstream that the wave plan is complete and that the metadata is ready. This communication should adhere to the governance you defined per the Project governance playbook for AWS large migrations.

At the end of this task, you have completed the following.

Output	Description
Wave plan	You have planned a wave, identified the servers, applications, and databases in that wave, and defined a start date and cutover date and time.
Source infrastructure metadata	You have collected the source infrastructure metadata, such as the server names and operating systems.
Target infrastructure metadata	You have collected the target infrastructure metadata, such as the target subnets, security groups, and AWS account.
Notification completed	You have notified the migration workstream that the wave plan and metadata are ready.

The portfolio team repeats all three tasks in this stage for each sprint until the migration project is completed.

Resources

AWS large migrations

Strategy

- AWS large-migration strategy and best practices

Guide

- Guide for AWS large migrations

Playbooks

- Foundation playbook for AWS large migrations
- Project governance playbook for AWS large migrations
- Migration playbook for AWS large migrations

Additional references

Tools and services

- AWS CloudEndure Migration Factory Solution
- Free Cloud Migration Services on AWS
- AWS Database Migration Service
- Migrate with AWS
- Flexera One Cloud Migration and Modernization (Flexera website)
- TDS TransitionManager (TDS website)

AWS Prescriptive Guidance

- Automating large-scale server migrations with CloudEndure Migration Factory
- Best practices for assessing applications to be retired during a migration to the AWS Cloud
- Evaluating migration readiness
- Get started with automated portfolio discovery
- Mobilize your organization to accelerate large-scale migrations
- Migration strategy for relational databases
- Application portfolio assessment guide for AWS Cloud migration

Videos

- Executing a large-scale migration to AWS (AWS re:Invent 2020)
- CloudEndure Migration Factory best practices (AWS re:Invent 2020)

AWS Prescriptive Guidance glossary

AI and ML terms

The following are commonly used terms in artificial intelligence (AI) and machine learning (ML)-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

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?"
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.
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.
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.
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.
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.
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.
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?"
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).
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.
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.
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.
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.

Migration terms

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

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. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on 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.
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 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.
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 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.
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.
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.
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 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.
configuration management database (CMDB)	A database that contains information about a company's hardware and software products, configurations, and inter-dependencies. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.
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.
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.
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.
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.
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.
large migration	A migration of 300 or more servers.
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 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 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 CloudEndure 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 strategy	The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs (p. 56) entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.
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).
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.

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.
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.
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.
responsible, accountable, consulted, informed (RACI) matrix	A matrix that defines and assigns roles and responsibilities in a project. For example, you can create a RACI to define security control ownership or to identify roles and responsibilities for specific tasks in a migration project.
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.
service-level agreement (SLA)	An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.
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.
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.
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.

Modernization terms

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

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.
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.
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.
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.
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.
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.
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.
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. For more information, see the Two-pizza team section of the Introduction to DevOps on AWS whitepaper.

Contributors

The following individuals contributed to this document:

- Pratik Chunawala, Senior Cloud Architect, Amazon Web Services
- Rodolfo Jr. Cerrada, Senior Application Architect, Amazon Web Services
- Dwayne Bordelon, Senior Cloud Application Architect, Amazon Web Services
- Wally Lu, Principal Consultant, Amazon Web Services

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.

update-history-change	update-history-description	update-history-date
Initial publication (p. 64)	—	February 28, 2022