AWS Prescriptive Guidance

Automated patching for mutable instances in the hybrid cloud using AWS Systems Manager

AWS Prescriptive Guidance: Automated patching for mutable instances in the hybrid cloud using AWS Systems Manager

Table of Contents

Introduction 1
Overview 2
    Terms and concepts 2
    Key user stories 3
Patching process 5
Design for mutable EC2 instances 7
    Automated process 7
Design for multiple AWS accounts and Regions 9
    Automated process 9
    Architectural considerations and limitations 10
        Maintenance window quotas per account 10
        Other considerations 11
Design for on-premises instances in hybrid cloud environment 12
    Automated process 12
    Architectural considerations and limitations 13
Key stakeholders, roles, and responsibilities 15
    User personas 15
    RACI matrix 16
Next steps 18
Additional resources 19
AWS Prescriptive Guidance glossary 20
Document history 28

Automated patching for mutable instances in the hybrid cloud using AWS Systems Manager

Chandra Allaka, Senior Consultant, Operations Integration, AWS Professional Services

June 2020

This prescriptive guide describes an automated patching solution that uses Amazon Web Services (AWS) Systems Manager. You can use this solution to patch both your mutable (long-running) Amazon Elastic Compute Cloud (Amazon EC2) instances that span multiple AWS accounts and AWS Regions, and your on-premises instances.

This guide is for users who are involved in designing and building operational capabilities in a hybrid cloud environment to enable application teams to comply with their enterprise's patch policies. It provides you with a self-service mechanism to deploy pre-approved patches to your application servers.

This guide assumes a good understanding of the following AWS services and concepts:

- Systems Manager – Provides a unified user interface for viewing operational data from multiple AWS services and automating operational tasks across your AWS resources.
- Systems Manager Inventory – Provides visibility into your Amazon EC2 and on-premises computing environment. You can use Inventory to collect metadata from your managed instances.
- Systems Manager Patch Manager – Automates the process of patching managed instances with security-related and other types of updates.
- Systems Manager Maintenance Windows – Let you define a schedule for performing potentially disruptive actions on your instances, such as patching an operating system, updating drivers, or installing software or patches.
- AWS Lambda – Lets you run code without provisioning or managing servers.
- Amazon QuickSight – Lets you easily create and publish interactive dashboards, including machine learning (ML) Insights. You can access dashboards from any device and embed them into your applications, portals, and websites.
- Tagging – Lets you assign metadata to your AWS resources in the form of tags. Each tag is a label consisting of a user-defined key and value. Tags can help you manage, identify, organize, search for, and filter resources.

Patch management overview

If you are involved in application or infrastructure operations, you understand the importance of an operating system (OS) patching solution that is flexible and scalable enough to meet the varied requirements from your application teams. In a typical organization, some application teams use an architecture that involves immutable instances whereas others deploy their applications on mutable instances.

Immutable instance patching involves applying the patches to the Amazon Machine Images (AMIs) that are used to provision the immutable EC2 application instances. Mutable instance patching involves an in-place patch deployment to running instances during a scheduled maintenance window.

This prescriptive guide describes how you can use AWS Systems Manager Patch Manager to patch mutable instances that span multiple AWS accounts and AWS Regions in an automated way, based on the maintenance windows and patch groups defined by the application teams on their servers through tags.

The guide describes an automated patching solution that uses AWS Lambda to automate patching configurations and scheduling, using Patch Manager and maintenance windows. Amazon QuickSight provides the necessary reporting and dashboard capabilities to report on patch compliance.

In addition, this guide describes a reference architecture for hybrid cloud environments. Users who run their applications in a hybrid cloud setup look for opportunities to consolidate, simplify, standardize, and optimize their patch management operations across AWS and their on-premises infrastructure. The guide explains how the automated patching solution for mutable instances can be extended to support hybrid cloud scenarios.

This guide describes:

- Key user stories for patch management
- The patching process
- Patch management for mutable instances in a single account and single AWS Region; architectural considerations and limitations
- Patch management for mutable instances in a multi-account, multi-Region environment; architectural considerations and limitations
- Patch management for on-premises instances in a hybrid cloud environment; architectural considerations and limitations
- Key stakeholders, roles, and responsibilities

> **Note**
> This guide describes an architecture for an automated solution (referred to as the *automated patching solution*) that you can implement to support your patch management requirements for mutable instances. It doesn't provide the code for building the solution.

Terms and concepts

Term	Definition
Immutable instances	Immutable instances are EC2 server instances that do not undergo any changes while they're

Term	Definition
	running. If changes are required, you create a new instance with the updated server image, redeploy the instance, and destroy the existing server image.
Patch baseline	A patch baseline is specific to an OS type and defines the patch list approved for installation on the instances. For more information, see About predefined and custom patch baselines in the Systems Manager documentation.
Patch group	A patch group represents the servers within an application environment that are targets of a specific patch baseline. Patch groups help ensure that the right patch baselines are deployed to the correct set of instances. They also help avoid deploying patches before they have been adequately tested. Patch groups are represented by the **Patch Group** tag. For more information, see About patch groups in the Systems Manager documentation.
Maintenance window	Maintenance windows let you define a schedule for performing potentially disruptive actions on instances, such as patching an operating system, updating drivers, or installing software or patches. Each maintenance window has a schedule, a maximum duration, a set of registered target instances, and a set of registered tasks. Patch groups are represented by the **Maintenance Window** tag. For more information, see About patching schedules using maintenance windows in the Systems Manager documentation.

Key user stories

The typical OS patching process involves three tasks:

1. Scanning the EC2 instances and the on-premises servers for applicable OS patches.
2. Grouping and patching the instances at a suitable time.
3. Reporting patching compliance across the server environment.

The following table lists the key user stories for patch management.

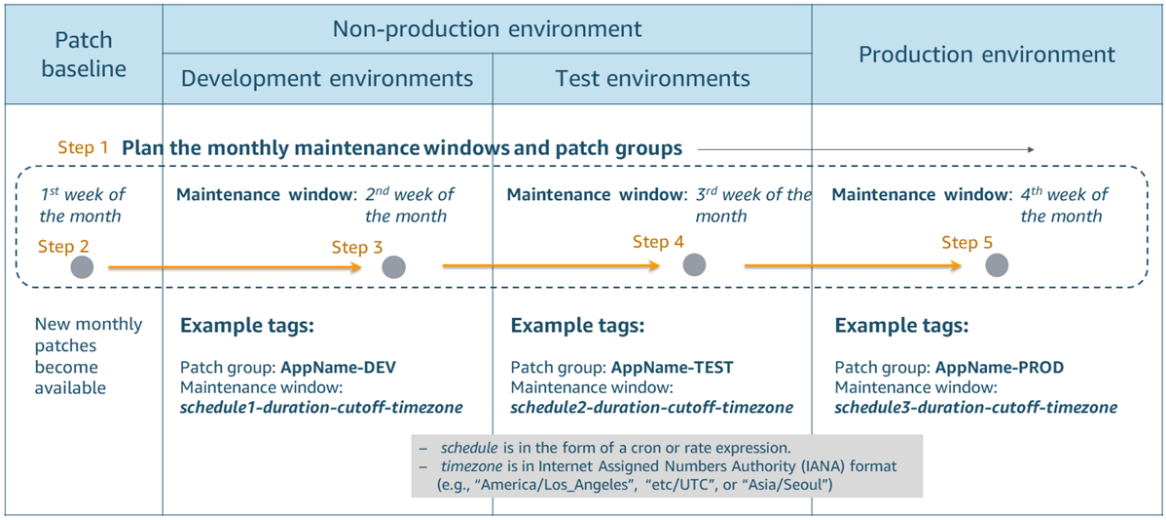
Scenario	User role	Description
Patching mechanism	Application dev/support teams	As an application team member who is responsible for OS patching, I need a mechanism to patch my long-running or mutable instances, so I can mitigate any OS security

Scenario	User role	Description
		vulnerabilities and also ensure that the instances comply with the patching baseline defined by the security team.
Patching solution	Cloud service owner	As a cloud service owner who is responsible for providing cloud services to the application teams, I need to build an OS patching solution that supports multiple AWS accounts and AWS Regions as well as on-premises servers, so application teams can mitigate any OS security vulnerabilities and also stay compliant with the patching baseline defined by the security team.
Patching compliance reporting	Security operations manager	As a security operations manager who is responsible for ensuring patch compliance, I need detailed patch compliance reporting and information across the cloud landscape, so I can identify servers that are not compliant with the patch baseline and alert teams to implement the mitigation required.
Definition of roles and responsibilities	Cloud service owner	As a cloud service owner, I need to build a well-defined roles and responsibilities matrix that explains who does what in managing the hybrid cloud patching solution I built, so obligations for patching operations are published and met.

Patching process

The primary users of the patching solution are the application development and operations teams. Each application is typically deployed into multiple environments, such as development, test (integration, user acceptance, and so on), and production. The application teams have to plan the patching schedules for each environment, so when a patch is applied to the production environment, it has already been tested and determined to have no adverse effects on the application.

The following workflow provides an example of how you can plan patching windows for an application that is deployed in multiple environments and how to configure tags.



- **Step 1.** Each application team plans their maintenance windows for their servers within various environments, and sets up the tags that represent the servers' patch groups and maintenance windows accordingly:
  - The **Patch Group** tag represents the servers within an application environment that are the targets of a specific patch baseline. Patch groups help ensure that the right patch baselines are deployed to the correct set of instances. Patch groups also help avoid deploying patches in the production environment before they have been adequately tested.
  - If the application servers include multiple operating systems, the application team creates patch groups based on the combination of the environment and operating system. A patch group can be registered with only one patch baseline, and an instance can be part of only one patch group.

    For example: *appname*`-DEV-WIN` and *appname*`-DEV-RHEL`
  - The **Maintenance Window** tag represents the schedule for patching the servers. **All servers in a patch group should be in the same maintenance window.** The maintenance window tag should follow a consistent format for cron and rate expressions so that a Lambda function that you define can parse the expressions easily. (In this guide, we'll refer to this Lambda function as `automate-patch`.)

    For example: *`schedule-duration-cutoff-timezone`*

    `cron(0 2 ? * SAT#3 *)` represents 02:00 AM on the third Saturday of every month. For detailed information about cron and rate expressions, see the Systems Manager documentation.
- **Step 2.** Systems Manager Patch Manager makes new patches available regularly through operating system-specific patch baselines based on the configurations defined.

    - For each operating system, you can define a custom patch baseline that includes the approval rules and the patches that need to be applied to the instances across the cloud environment.
- **Step 3.** Your custom automation code configures Patch Manager to set up patching based on the **Patch Group** and **Maintenance Window** tags, and applies the patches to the development environment.
    - After patching is complete, the application development and support teams test the application and verify that everything works correctly.
    - If the application encounters any problems with the new patch, application teams ask the cloud services team to halt patching to other patch groups and other environments, by either disabling the maintenance windows or deregistering the patch task execution.
- **Step 4.** After the development environment is patched successfully, patching is deployed to any other non-production environments. As with the development environment, the application is tested and verified to be working correctly in all non-production environments. If there are any issues, application teams ask the cloud services team to halt patching to the production environment.
- **Step 5.** After all the non-production environments have been patched successfully, patching is applied to the production environment.

Patching solution design for mutable EC2 instances

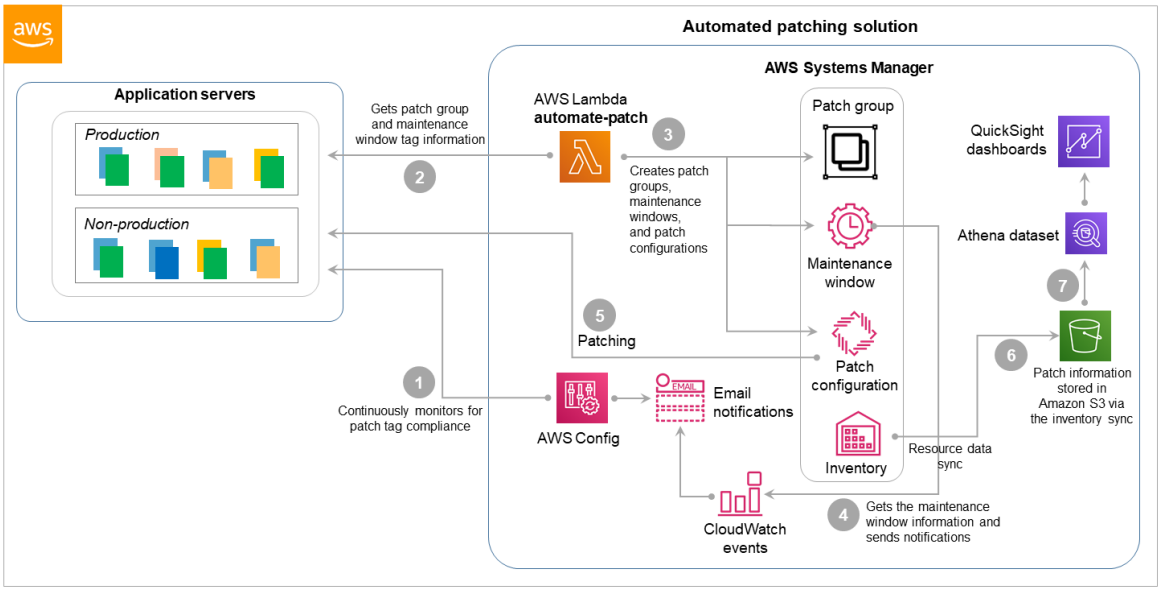
The patching process for mutable instances involves the following teams and actions:

- The **application (DevOps) teams** define the patch groups for their servers based on application environment, OS type, or other criteria. They also define the maintenance windows specific to each patch group. This information is stored on the **Patch Group** and **Maintenance Window** tags of the EC2 application instances. During each patch cycle, the application teams prepare for patching, test the application after patching, and troubleshoot any issues with their applications and OS during patching.
- The **security operations team** defines the patch baselines for various OS types that are used by the application teams, approve the patches, and make the patches available through Systems Manager Patch Manager.
- The **automated patching solution** runs on a regular basis and deploys the patches defined in the patch baselines based on the user-defined patch groups and maintenance windows. The patch compliance information is obtained through a resource data sync in Systems Manager Inventory, and is used for patch compliance reporting through Amazon QuickSight dashboards.
- The **governance and compliance teams** define the patching guidelines, define exception processes and mechanisms, and obtain the compliance reporting from Amazon QuickSight.

For detailed information about the key stakeholders involved in a successful OS patch management solution and their responsibilities, see the Key stakeholders, roles, and responsibilities (p. 15) section later in this guide.

Automated process

The automated patching solution uses multiple AWS services that work in tandem to deploy the patches to the EC2 instances. This process involves AWS Config, AWS Lambda, Systems Manager, Amazon Simple Storage Service (Amazon S3), and Amazon QuickSight. The following diagram shows the reference architecture and workflow.



The workflow includes these steps, where the step numbers match the callouts in the diagram:

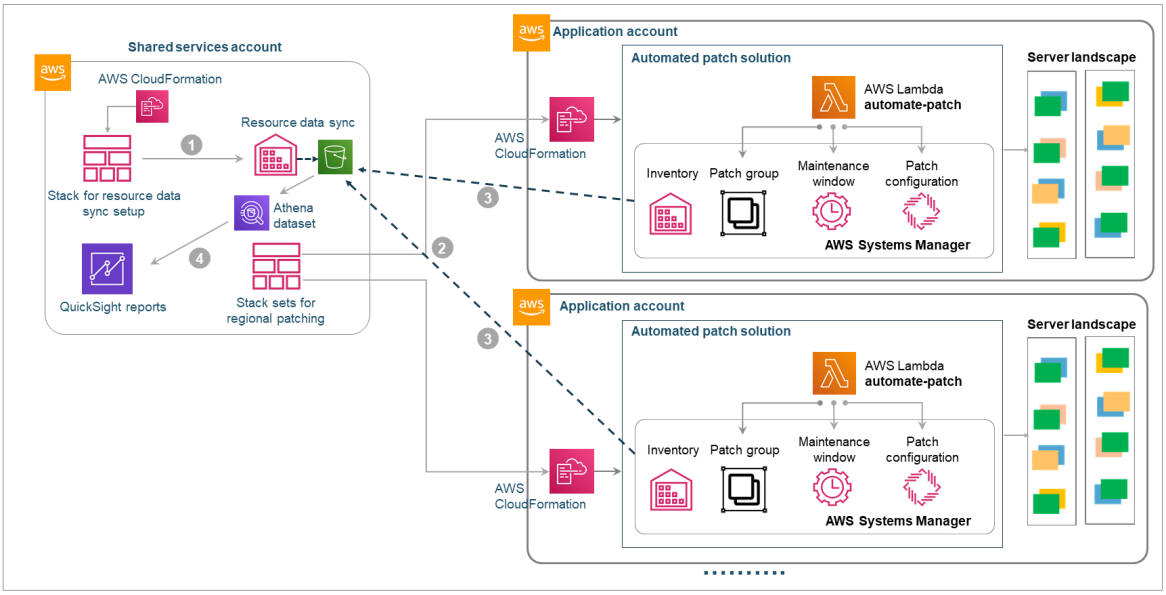
1. AWS Config continuously monitors for the following and sends notifications with the details of non-compliant instances and the configurations needed:
   - Patch tagging compliance on EC2 instances. AWS Config checks for instances that don't have **Patch Group** and **Maintenance Window** tags.
   - The AWS Identity and Access Management (IAM) instance profile with the Systems Manager role, which allows Systems Manager to manage the instances.
2. The Lambda function (we'll call it `automate-patch`) runs on a predefined schedule and collects the **Patch Group** and **Maintenance Window** information for all the servers.
3. The `automate-patch` function then creates or updates the appropriate patch groups and maintenance windows, associates the patch groups with the patch baselines, configures the patch scan, and deploys the patching task. Optionally, the `automate-patch` function also creates events in Amazon CloudWatch Events to notify users of impending patches.
4. Based on the maintenance windows, the events send patch notifications to the application teams with the details of the impending patching operation.
5. Patch Manager performs system patching based on the defined schedule and the patch groups.
6. A resource data sync in Systems Manager Inventory gathers the patching details and publishes them to an S3 bucket.
7. Patch compliance reporting and dashboards are built in Amazon QuickSight from the S3 bucket information.

Patching solution design for multiple AWS accounts and Regions

You can extend the automated patching solution to support servers that span multiple AWS accounts and multiple AWS Regions. The extended solution involves setting up the patch automation solution in each AWS account through AWS CloudFormation StackSets in a shared services account, and configuring a resource data sync across the accounts with the shared services account.

Automated process

The following diagram illustrates the architecture for this scenario. This architecture includes AWS CloudFormation StackSets and an AWS shared service account.



The workflow is similar to the process described in the previous section, but involves the following additional steps, where the step numbers match the callouts in the diagram:

1. In the shared services account, an AWS CloudFormation stack set is used to configure the S3 bucket for resource data sync through Systems Manager Inventory.
2. The CloudFormation stack set creates the stack with the `automate-patch` Lambda function, sets up the patch baselines, and sets up Systems Manager Inventory resource data sync on the application accounts, to synchronize resources in the shared services account.
3. The resource information in the application accounts is synchronized with the resource information in the shared services account.
4. Amazon QuickSight generates patch compliance reports, using the Amazon Athena dataset for the synchronized resource information.

Architectural considerations and limitations

Maintenance window quotas per account

The architecture illustrated and described in the previous section creates a maintenance window for each patch group. However, the quota for the number of maintenance windows per AWS account is 50 (assuming that you haven't requested a service quota increase). If you expect the number of patch groups to exceed 50 groups in a single AWS account, this architecture won't scale to meet your requirements.

If a service quota increase isn't sufficient for your needs, there are two options for managing this challenge: using predefined maintenance windows and using CloudWatch Events. Here are the advantages and disadvantages of each approach.

Option 1. Use predefined maintenance windows

- Define a list of maintenance windows with various time windows (for example, 15 to 20 maintenance windows per account).
- The application teams choose the maintenance windows that suit them from the predefined list and tag the instances accordingly.
- Update the automated patching solution to map the patch groups to the selected maintenance windows instead of creating new maintenance windows.

Pros:

- Simplified management.

Cons:

- Less flexibility for defining custom maintenance windows.
- When multiple patch groups share maintenance windows and patch tasks, canceling a specific patch task for a specific patch group requires additional manual effort.

Option 2. Use CloudWatch Events to trigger patch tasks instead of using maintenance windows

- Instead of creating maintenance windows, use CloudWatch Events to trigger patch tasks based on the schedule and the patch groups.
- In this scenario, each patch group is associated with a CloudWatch Events event instead of a maintenance window.
- Update the automated patching solution to create events instead of maintenance windows.

Pros:

- Scalable design.
- Provides flexibility for defining custom maintenance windows.

Cons:

- Maintenance windows provide additional functionality (such as duration and cutoff times) that aren't available with CloudWatch Events.

Other considerations

- The automated patching solution described in this section doesn't support EC2 instances that are shut down.
- This process supports EC2 instances in public subnets. To patch instances in private subnets, you must deploy a local patch repository like Windows Server Update Services (WSUS).
- You must adjust the frequency for running the Lambda function so patch groups and maintenance windows are updated according to your required schedule.

Patching solution design for on-premises instances in a hybrid cloud environment

You can also extend the solution described in this guide to patch on-premises server instances in a hybrid cloud environment.

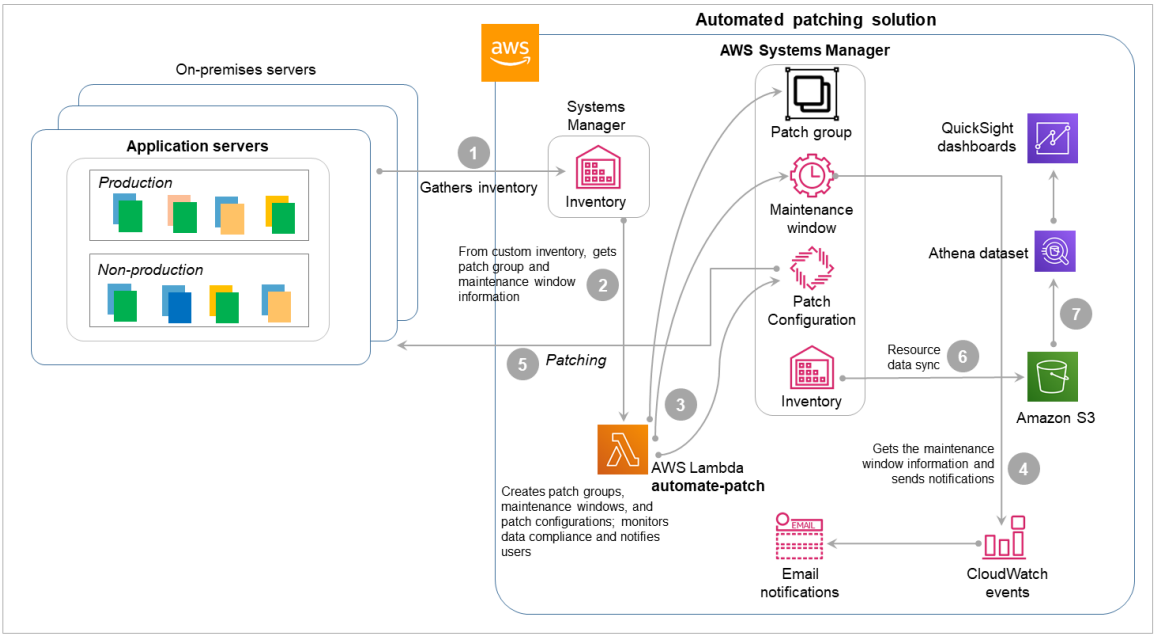
The standard patching process for on-premises instances consists of two steps:

- You configure your on-premises servers to be managed by Systems Manager. For the details of this process, see Setting up Systems Manager for hybrid environments in the Systems Manager documentation.
- You configure the appropriate **Patch Group** and **Maintenance Window** tags for these on-premises managed instances by using the AWS Command Line Interface (AWS CLI) add-tags-to-resource command.

However, this approach requires either the application team or the cloud team to manually run the AWS CLI commands whenever they want to perform changes to the patch groups or maintenance windows.

Automated process

The following illustration describes an alternate approach to patching on-premises instances that uses the Systems Manager custom inventory option. This process is an extension of the automated patching solution that we described earlier for mutable EC2 instances.



1. Instead of using tags, Systems Manager captures the patch information (patch groups and maintenance windows) from the on-premises managed instances through a custom inventory collection.

```
Sample custom inventory JSON file
{
    "SchemaVersion": "1.0",
    "TypeName": "Custom:PatchInformation",
    "Content": {
        "Patch Group": "<APP-PROD>",
        "Maintenance Window": "XXX"
    }
}
```

2. The Lambda `automate-patch` function runs every day, collects the patch group and maintenance window information from the on-premises server custom inventory, and creates the **Patch Group** and **Maintenance Window** tags on the managed instances.
3. The Lambda `automate-patch` function then creates or updates the appropriate patch groups and maintenance windows, associates the patch groups with the patch baselines, configures the patch scans, and deploys the patching task, based on the custom inventory that was gathered. Optionally, the `automate-patch` function also creates events in CloudWatch Events to notify users of impending patches.
4. Based on the maintenance windows, the events send patch notifications to the application teams with the details of the impending patching operation.
5. Patch Manager performs system patching based on the defined schedule and the patch groups.
6. A resource data sync in Systems Manager Inventory gathers the patching details and publishes them to an S3 bucket.
7. Patch compliance reporting and dashboards are built in Amazon QuickSight from the S3 bucket information.

Architectural considerations and limitations

As discussed in the previous sections, there are two approaches to patching on-premises instances: through custom inventory or by using tags. Here are the advantages and disadvantages of each approach.

Option 1. Use custom inventory for patch information

- Application teams working with on-premises servers configure the patch information in the custom inventory file, and Systems Manager picks that information.
- The custom inventory patch information is then used to create the patch tasks.

Pros:

- Much simpler to configure because it involves only a file update.

Cons:

- The changes to the patch configuration are limited to the inventory collection schedule.

Option 2. Use tags for on-premises managed instances

- Application teams working with on-premises servers create **Patch Group** and **Maintenance Window** tags by using AWS CLI with the appropriate patch information.

- The tag information is used to create the patch tasks.

Pros:

- Consistent approach across AWS and on premises to drive patching standardization and automation.

Cons:

- Application teams working with on-premises instances have to learn and use AWS CLI to create or update the tags.

Key stakeholders, roles, and responsibilities in patch management

Successful OS patch management requires having well-defined roles and responsibilities for supporting your automated patching solution and optimizing it continually. This section describes suggested roles and responsibilities that you can modify according to your needs and organizational structure.

User personas

The following table describes the user personas involved with the automated patching solution.

User persona	Description
Consumers (C)	The patch management solution for long-running instances is used by different teams involved in OS management, including: • Development teams that manage full-stack application environments. • Operations teams that manage the application server OS.
Cloud engineering (CE)	The team that's responsible for: • Continuously optimizing the patch management solution. • Building cloud services automation. • Supporting the automation.
Cloud business office (CBO)	The team that's involved in: • Managing the consumer experience for the solution. • Enablement and user engagement. • Making sure that the patch solution meets consumers' needs.
Cloud service/product owner (CPO)	The person who is responsible for: • Providing cloud services to consumers. • Working closely with the leadership team to align the services delivery with expectations and guidelines. • Managing all customer expectations and escalations related to the platform. • Owning the platform roadmap.

User persona	Description
Security operations (SO)	The team that manages patch baselines and approvals.
Security operations manager (SOM)	The manager who is responsible for patch compliance.

RACI matrix

The following responsible, accountable, consulted, informed (RACI) matrix specifies the activities involved with the patch management solution. For each step in the process, it lists the stakeholders and their involvement:

- **R** – responsible for completing the step
- **A** – accountable for approving and signing off on the work
- **C** – consulted to provide input for a task
- **I** – informed of progress, but not directly involved in the task

Patch management solution	CPO	CBO	CE	SO	SOM	C
Patch management product roadmap execution	A	C	R	C	C	I
Patch management architecture and design	A	I	R	C	I	
Patch management development and configuration	A		R	C		
Patch management validation and testing	A	I	R	I	I	
New AWS account, application, and server onboarding for patching	A	C	R	I		
User engagement	A	R	I	I	I	

Patch management solution	CPO	CBO	CE	SO	SOM	C
and enablement						
User feedback and escalation management	A	R		I	I	
Product change management	A	R	C	I		
Issue management and resolution	A		R	C		
Server patching and patch compliance			C	C		AR
Patch baseline configuration			C	R	A	C
Patch reporting and compliance			C	R	AR	I

Next steps

This guide has described an automated patching solution for mutable instances on AWS and on-premises instances in a hybrid cloud environment. To build the solution, we recommend that you consult the documentation for the AWS services described in this guide. If you have questions, please get in touch with your AWS account team for assistance.

For more information, see the Additional resources (p. 19) section.

Additional resources

AWS resources

- AWS Prescriptive Guidance
- AWS documentation
- AWS general reference
- AWS glossary

AWS services

- AWS CloudFormation
- Amazon CloudWatch
- Amazon EC2
- IAM
- AWS Lambda
- Amazon QuickSight
- AWS Systems Manager

Other resources

- How to patch Amazon EC2 instances in private subnets Using AWS Systems Manager (AWS Management & Governance blog)
- How Moody's uses AWS Systems Manager to patch servers across multiple cloud providers (AWS Management & Governance blog)
- Setting up AWS Systems Manager for hybrid environments (Systems Manager documentation)
- Centralized multi-account and multi-Region patching with AWS Systems Manager Automation (AWS Management & Governance blog)
- Patching your Amazon EC2 instances using AWS Systems Manager Patch Manager (AWS Management & Governance blog)
- How to Patch, Inspect, and Protect Microsoft Windows Workloads on AWS—Part 1 (AWS Security blog)

AWS Prescriptive Guidance glossary

AI and ML terms (p. 20) | Migration terms (p. 21) | Modernization terms (p. 25)

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. 21) 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.

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. 28)	—	June 12, 2020