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

# AWS Prescriptive Guidance

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

# Table of Contents

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

*Chandra Allaka, Amazon Web Services (AWS)*

*June 2020* (document history)

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 Quick – 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 Quick 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 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 |

| Term | Definition |
| --- | --- |
| | 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. Maintenance windows 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 |

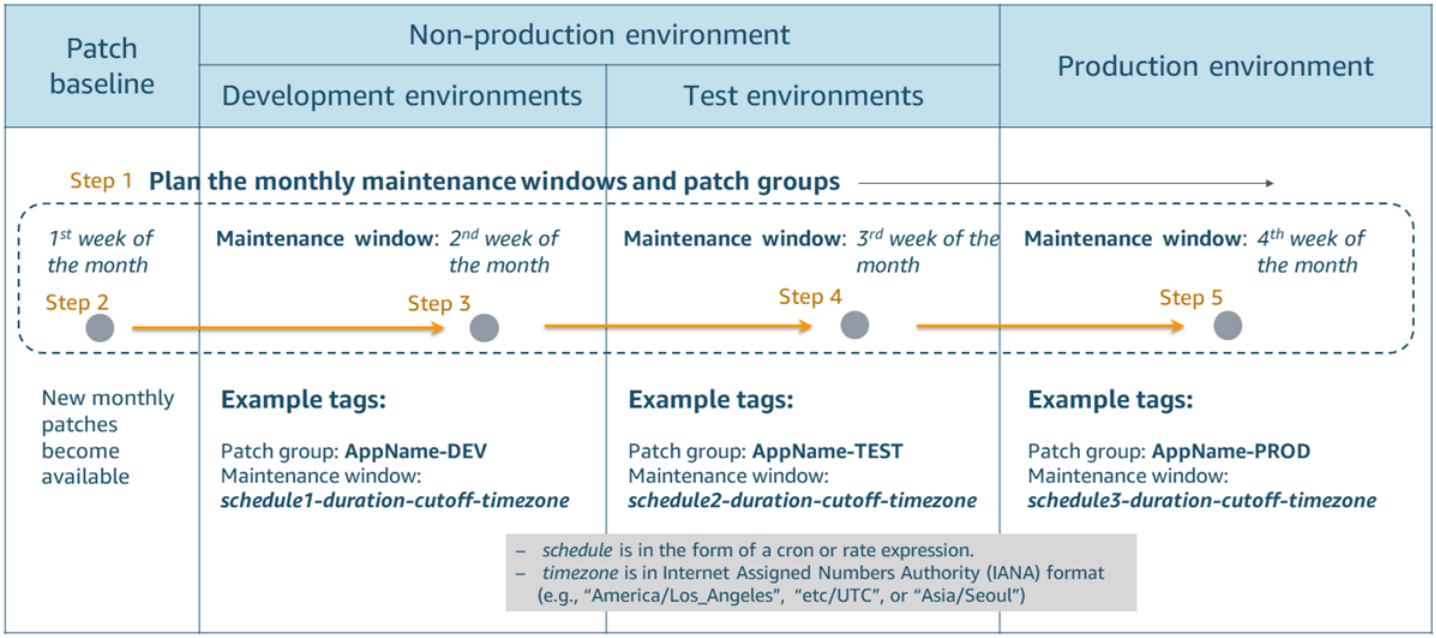
| Scenario | User role | Description |
| --- | --- | --- |
| | | any OS security vulnerabilities and also ensure that the instances comply with the patching baseline defined by the security team. |
| **Patching solution** | Cloud service owner | As a cloud service owner who is responsible for providing 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. |

| Scenario | User role | Description |
| --- | --- | --- |
| **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

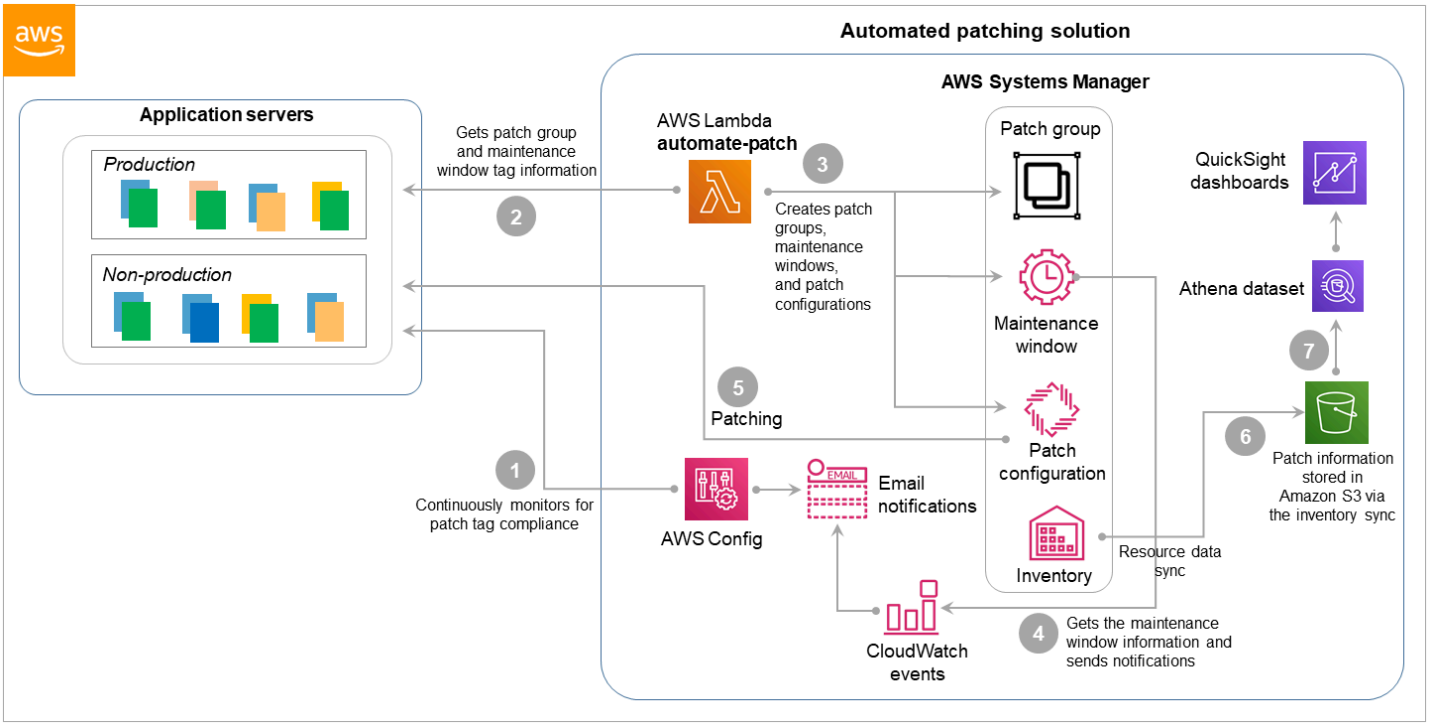
The patching process for mutable instances involves the following teams and actions:

- 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 Quick dashboards.
- The **governance and compliance teams** define the patching guidelines, define exception processes and mechanisms, and obtain the compliance reporting from Quick.

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 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 Quick. 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.

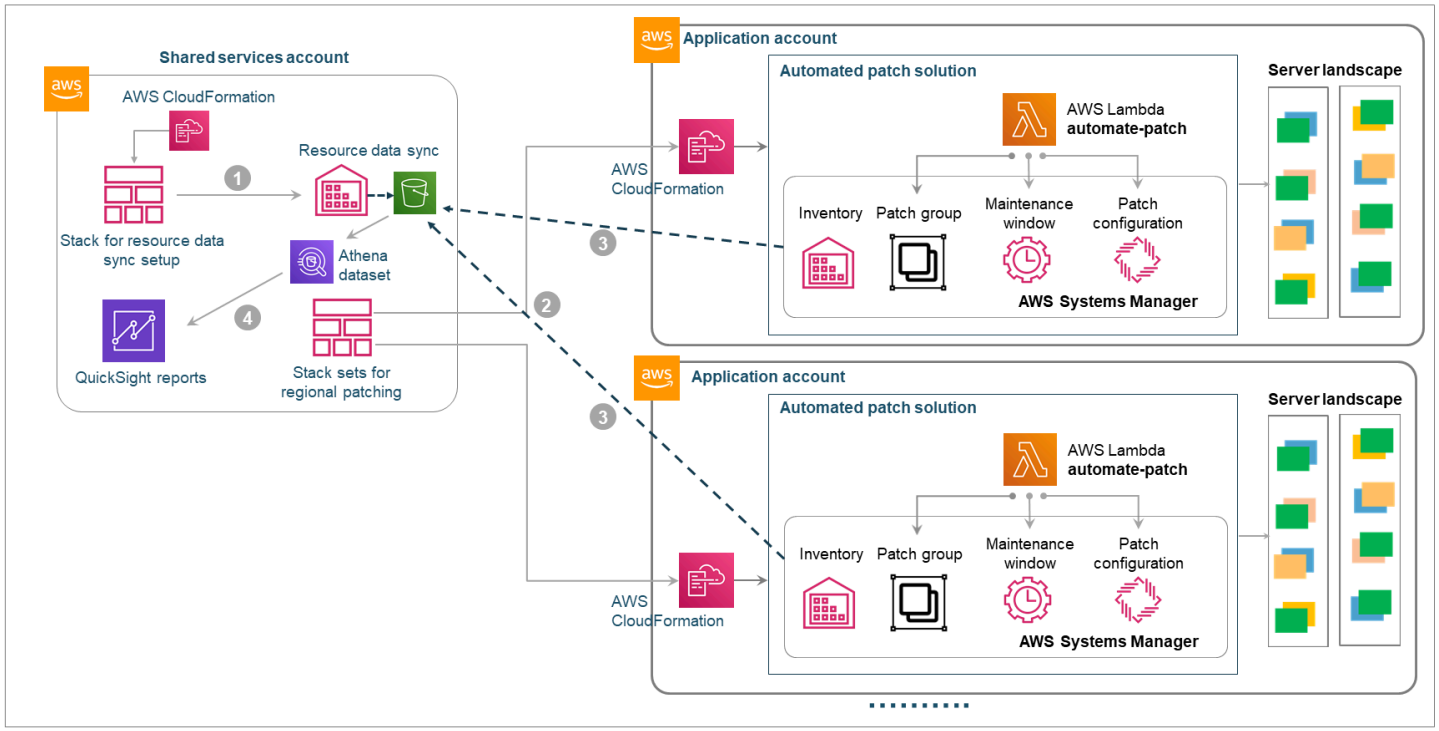
7. Patch compliance reporting and dashboards are built in Quick from the S3 bucket information.

# 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 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 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. Quick 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.

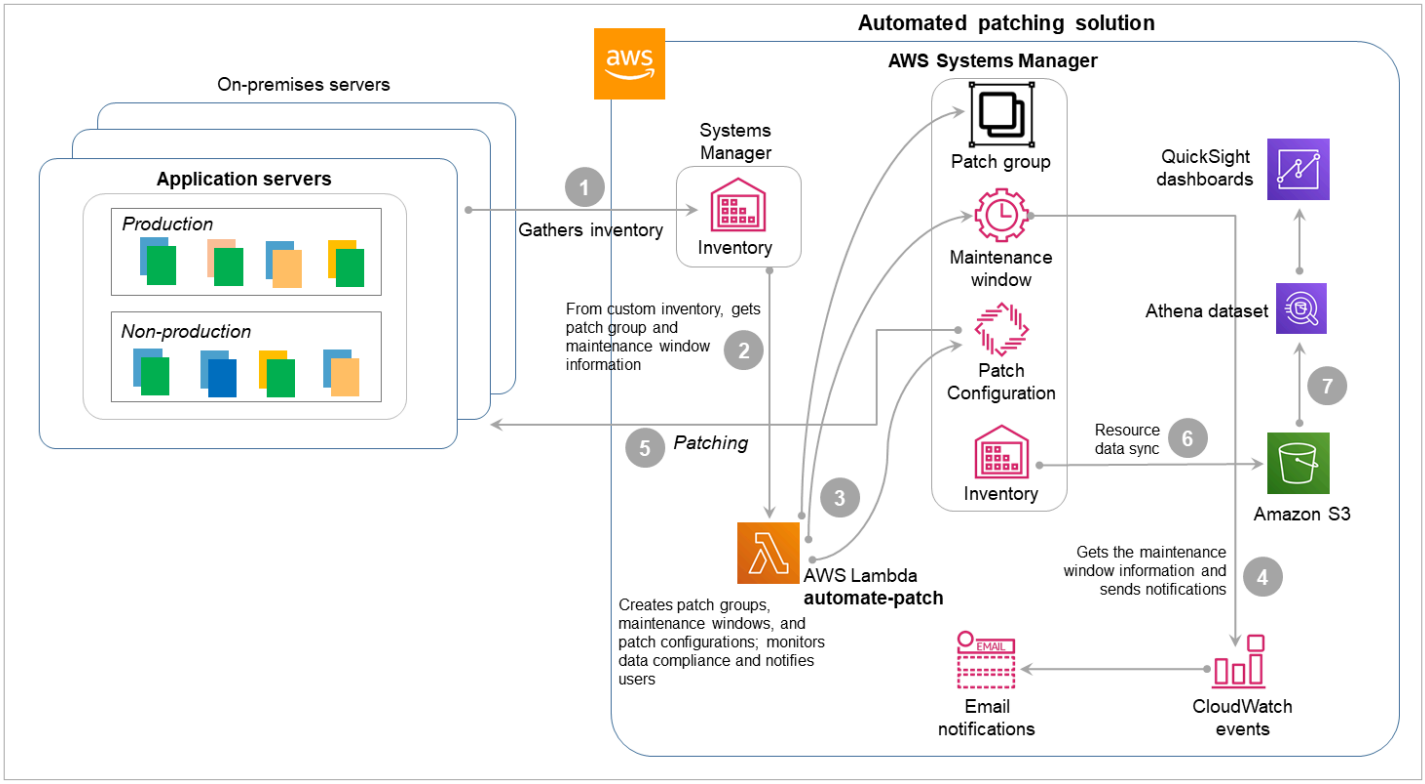
The standard patching process for on-premises instances consists of two steps:

- 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 Quick 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:<br>• Development teams that manage full-stack application environments.<br>• Operations teams that manage the applicati on server OS. |
| **Cloud engineering (CE)** | The team that's responsible for:<br>• Continuously optimizing the patch management solution.<br>• Building cloud services automation.<br>• Supporting the automation. |
| **Cloud business office (CBO)** | The team that's involved in:<br>• Managing the consumer experience for the solution.<br>• Enablement and user engagement. |

| User persona | Description |
| --- | --- |
| | • Making sure that the patch solution meets consumers' needs. |
| **Cloud service/product owner (CPO)** | The person who is responsible for:<br>• Providing cloud services to consumers.<br>• Working closely with the leadership team to align the services delivery with expectations and guidelines.<br>• Managing all customer expectations and escalations related to the platform.<br>• Owning the platform roadmap. |
| **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 onboardin** | A | C | R | I | | |

| Patch management solution | CPO | CBO | CE | SO | SOM | C |
|---|---|---|---|---|---|---|
| **g for patching** | | | | | | |
| **User engagement and enablement** | A | R | I | I | I | |
| **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 management solution | CPO | CBO | CE | SO | SOM | C |
|---|---|---|---|---|---|---|
| **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 section.

# Additional resources

### AWS resources

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

### AWS services

- CloudFormation
- Amazon CloudWatch
- Amazon EC2
- IAM
- AWS Lambda
- Amazon Quick
- 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)

# Document history

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

| Change | Description | Date |
| --- | --- | --- |
| Initial publication | — | June 12, 2020 |