**Design Approach for Automating the Deployment: Lambda+API+S3**

**Prerequisites Services:**

**AWS CloudFormation, AWS CLI, S3, API, AWS Lambada, System Manger Ansible, Jenkins, GitHub or GitLab, JiraTool**

**1.Lambda:**

AWS Lambda is a flexible service designed for a wide variety of use-cases. Across the millions of AWS customers using Lambda every month, serverless applications generally fall into several common categories:

* **Web applications**: serve the front-end code via Amazon S3 and Amazon CloudFront, or automating the entire deployment and hosting with AWS Amplify Console.
* **Web and mobile backends**: the front-ends interact with the backend via API Gateway. Integrated authorization and authentication are provided by Amazon Cognito or APN Partners like Auth0.
* **Data processing**: event-based processing tasks triggered by data changes in data stores, or streaming data ETL tasks with Amazon Kinesis and Lambda.
* **Parallelized computing tasks**: splitting highly complex, long-lived computations to individual tasks across many Lambda function instances to process data more quickly in parallel.
* **Internet of Things (IoT) workloads**: processing data generated by physical IoT devices.

Additionally, many workloads are hybrid serverless applications, especially where legacy systems are being migrated from either on-premises or instance-based environments. In this case, developers can gradually migrate functionality from a legacy system to a Lambda-based application.

**2.API reference**

This section contains the AWS Lambda API Reference documentation. When making the API calls, you will need to authenticate your request by providing a signature. AWS Lambda supports signature version 4. For more information, see [Signature Version 4 signing process](https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html) in the *Amazon Web Services General Reference*.

For an overview of the service, see [What is AWS Lambda?](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html).

You can use the AWS CLI to explore the AWS Lambda API. This guide provides several tutorials that use the AWS CLI.

**3.Certificate errors when using an SDK**

Because AWS SDKs use the CA certificates from your computer, changes to the certificates on the AWS servers can cause connection failures when you attempt to use an SDK. You can prevent these failures by keeping your computer's CA certificates and operating system up-to-date. If you encounter this issue in a corporate environment and do not manage your own computer, you might need to ask an administrator to assist with the update process. The following list shows minimum operating system and Java versions:

* Microsoft Windows versions that have updates from January 2005 or later installed contain at least one of the required CAs in their trust list.
* Mac OS X 10.4 with Java for Mac OS X 10.4 Release 5 (February 2007), Mac OS X 10.5 (October 2007), and later versions contain at least one of the required CAs in their trust list.
* Red Hat Enterprise Linux 5 (March 2007), 6, and 7 and CentOS 5, 6, and 7 all contain at least one of the required CAs in their default trusted CA list.
* Java 1.4.2\_12 (May 2006), 5 Update 2 (March 2005), and all later versions, including Java 6 (December 2006), 7, and 8, contain at least one of the required CAs in their default trusted CA list.

When accessing the AWS Lambda management console or AWS Lambda API endpoints, whether through browsers or programmatically, you will need to ensure your client machines support any of the following CAs:

* Amazon Root CA 1
* Starfield Services Root Certificate Authority - G2
* Starfield Class 2 Certification Authority

Root certificates from the first two authorities are available from [Amazon trust services](https://www.amazontrust.com/repository/), but keeping your computer up-to-date is the more straightforward solution. To learn more about ACM-provided certificates, see [AWS Certificate Manager FAQs.](https://aws.amazon.com/certificate-manager/faqs/#certificates)

**4.Amazon S3**

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

**5.Amazon S3 REST API Introduction**

Welcome to the *Amazon Simple Storage Service API Reference*. This guide explains the Amazon Simple Storage Service (Amazon S3) application programming interface (API). It describes various API operations, related request and response structures, and error codes. The current version of the Amazon S3 API is 2006-03-01.

Amazon S3 supports the REST API.

**Note**

Support for SOAP over HTTP is deprecated, but it is still available over HTTPS. However, new Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Read the following about authentication and access control before going to specific API topics.

Requests to Amazon S3 can be authenticated or anonymous. Authenticated access requires credentials that AWS can use to authenticate your requests. When making REST API calls directly from your code, you create a signature using valid credentials and include the signature in your request. For information about various authentication methods and signature calculations, see [Authenticating Requests (AWS Signature Version 4)](https://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html).

Making REST API calls directly from your code can be cumbersome. It requires you to write the necessary code to calculate a valid signature to authenticate your requests. We recommend the following alternatives instead:

* Use the AWS SDKs to send your requests (see [Sample Code and Libraries](https://aws.amazon.com/code)). With this option, you don't need to write code to calculate a signature for request authentication because the SDK clients authenticate your requests by using access keys that you provide. Unless you have a good reason not to, you should always use the AWS SDKs.
* Use the AWS CLI to make Amazon S3 API calls. For information about setting up the AWS CLI and example Amazon S3 commands see the following topics:

[Set Up the AWS CLI](https://docs.aws.amazon.com/AmazonS3/latest/dev/setup-aws-cli.html) in the *Amazon Simple Storage Service User Guide*.

[Using Amazon S3 with the AWS Command Line Interface](https://docs.aws.amazon.com/cli/latest/userguide/cli-s3.html) in the *AWS Command Line Interface User Guide*.

If you'd like to make your own REST API calls instead of using one of the above alternatives, there are some things to keep in mind. The REST API uses standard HTTP headers and status codes, so standard browsers and toolkits work as expected. In some areas, we have added functionality to HTTP (for example, we added headers to support access control). In these cases, we have done our best to add the new functionality in a way that matches the style of standard HTTP usage. For more information about making requests, see [Making requests](https://docs.aws.amazon.com/AmazonS3/latest/userguide/MakingRequests.html) in the *Amazon Simple Storage Service User Guide*. For additional details about developing using REST APIs, see [Developing with Amazon S3 using the REST API](https://docs.aws.amazon.com/AmazonS3/latest/userguide/developing-rest-api.html) in the *Amazon Simple Storage Service User Guide*.

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon S3 resources. For example, you must have permissions to create an S3 bucket or get an object from your bucket. If you use root credentials of your AWS account, you have all the permissions. However, using root credentials is not recommended. Instead, we recommend that you create IAM users in your account and manage user permissions. For more information, see [Managing Access Permissions to Your Amazon S3 Resources](https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html) in the *Amazon Simple Storage Service User Guide*.

**6.What is the AWS Serverless Application Model (AWS SAM)?**

The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build [serverless applications](https://aws.amazon.com/serverless/) on AWS.

A **serverless application** is a combination of Lambda functions, event sources, and other resources that work together to perform tasks. Note that a serverless application is more than just a Lambda function—it can include additional resources such as APIs, databases, and event source mappings.

You can use AWS SAM to define your serverless applications. AWS SAM consists of the following components:

* **AWS SAM template specification**. You use this specification to define your serverless application. It provides you with a simple and clean syntax to describe the functions, APIs, permissions, configurations, and events that make up a serverless application. You use an AWS SAM template file to operate on a single, deployable, versioned entity that's your serverless application. For the full AWS SAM template specification, see [AWS Serverless Application Model (AWS SAM) specification](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification.html).

* **AWS SAM command line interface (AWS SAM CLI)**. You use this tool to build serverless applications that are defined by AWS SAM templates. The CLI provides commands that enable you to verify that AWS SAM template files are written according to the specification, invoke Lambda functions locally, step-through debug Lambda functions, package and deploy serverless applications to the AWS Cloud, and so on. For details about how to use the AWS SAM CLI, including the full AWS SAM CLI Command Reference, see [AWS SAM CLI command reference](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-reference.html#serverless-sam-cli).

This guide shows you how to use AWS SAM to define, test, and deploy a simple serverless application. It also provides an [example application](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-getting-started-hello-world.html) that you can download, test locally, and deploy to the AWS Cloud. You can use this example application as a starting point for developing your own serverless applications.

**7.Benefits of using AWS SAM**

Because AWS SAM integrates with other AWS services, creating serverless applications with AWS SAM provides the following benefits:

* **Single-deployment configuration**. AWS SAM makes it easy to organize related components and resources, and operate on a single stack. You can use AWS SAM to share configuration (such as memory and timeouts) between resources, and deploy all related resources together as a single, versioned entity.

* **Extension of AWS CloudFormation**. Because AWS SAM is an extension of AWS CloudFormation, you get the reliable deployment capabilities of AWS CloudFormation. You can define resources by using AWS CloudFormation in your AWS SAM template. Also, you can use the full suite of resources, intrinsic functions, and other template features that are available in AWS CloudFormation.

* **Built-in best practices**. You can use AWS SAM to define and deploy your infrastructure as config. This makes it possible for you to use and enforce best practices such as code reviews. Also, with a few lines of configuration, you can enable safe deployments through Code Deploy, and can enable tracing by using AWS X-Ray.

* **Local debugging and testing**. The AWS SAM CLI lets you locally build, test, and debug serverless applications that are defined by AWS SAM templates. The CLI provides a Lambda-like execution environment locally. It helps you catch issues upfront by providing parity with the actual Lambda execution environment. To step through and debug your code to understand what the code is doing, you can use AWS SAM with AWS toolkits like the [AWS Toolkit for JetBrains](https://docs.aws.amazon.com/toolkit-for-jetbrains/latest/userguide/), [AWS Toolkit for PyCharm](https://aws.amazon.com/pycharm/), [AWS Toolkit for IntelliJ](https://aws.amazon.com/intellij/), and [AWS Toolkit for Visual Studio Code](https://aws.amazon.com/visualstudiocode/). This tightens the feedback loop by making it possible for you to find and troubleshoot issues that you might run into in the cloud.

* **Deep integration with development tools**. You can use AWS SAM with a suite of AWS tools for building serverless applications. You can discover new applications in the [AWS Serverless Application Repository](https://docs.aws.amazon.com/serverlessrepo/latest/devguide/). For authoring, testing, and debugging AWS SAM–based serverless applications, you can use the [AWS Cloud9 IDE](https://docs.aws.amazon.com/cloud9/latest/user-guide/). To build a deployment pipeline for your serverless applications, you can use [CodeBuild](https://docs.aws.amazon.com/codebuild/latest/userguide/), [CodeDeploy](https://docs.aws.amazon.com/codedeploy/latest/userguide/), and [CodePipeline](https://docs.aws.amazon.com/codepipeline/latest/userguide/). You can also use [AWS CodeStar](https://docs.aws.amazon.com/codestar/latest/userguide/) to get started with a project structure, code repository, and a CI/CD pipeline that's automatically configured for you. To deploy your serverless application, you can use the [Jenkins plugin](https://plugins.jenkins.io/aws-sam/).

**8.Walkthrough: Creating associations that run Ansible playbooks**

You can create State Manager associations that run Ansible playbooks by using the AWS-ApplyAnsiblePlaybooks SSM document. State Manager is a capability of AWS Systems Manager. This document offers the following benefits for running playbooks:

* Support for running complex playbooks
* Support for downloading playbooks from GitHub and Amazon Simple Storage Service (Amazon S3)
* Support for compressed playbook structure
* Enhanced logging
* Ability to specify which playbook to run when playbooks are bundled

**Note**

Systems Manager includes two SSM documents that allow you to create State Manager associations that run Ansible playbooks: AWS-RunAnsiblePlaybook and AWS-ApplyAnsiblePlaybooks. The AWS-RunAnsiblePlaybook document is deprecated. It remains available in Systems Manager for legacy purposes. We recommend that you use the AWS-ApplyAnsiblePlaybooks document because of the enhancements described here.

Associations that run Ansible playbooks aren't supported on macOS.

**Support for running complex playbooks**

The AWS-ApplyAnsiblePlaybooks document supports bundled, complex playbooks because it copies the entire file structure to a local directory before executing the specified main playbook. You can provide source playbooks in Zip files or in a directory structure. The Zip file or directory can be stored in GitHub or Amazon S3.

**Support for downloading playbooks from GitHub**

The AWS-ApplyAnsiblePlaybooks document uses the aws:downloadContent plugin to download playbook files. Files can be stored in GitHub in a single file or as a combined set of playbook files. To download content from GitHub, specify information about your GitHub repository in JSON format. Here is an example.

{

"owner":"*TestUser*",

"repository":"*GitHubTest*",

"path":"*scripts/python/test-script*",

"getOptions":"*branch:master*",

"tokenInfo":"{{*ssm-secure:secure-string-token*}}"

}

**Support for downloading playbooks from Amazon S3**

You can also store and download Ansible playbooks in Amazon S3 as either a single .zip file or a directory structure. To download content from Amazon S3, specify the path to the file. Here are two examples.

**Example 1: Download a specific playbook file**

{

"path":"https://s3.amazonaws.com/*doc-example-bucket/playbook.yml*"

}

**Example 2: Download the contents of a directory**

{

"path":"https://s3.amazonaws.com/*doc-example-bucket/ansible/webservers/*"

}

**Important**

If you specify Amazon S3, then the AWS Identity and Access Management (IAM) instance profile on your managed instances must be configured with the AmazonS3ReadOnlyAccess policy. For more information, see [Create an IAM instance profile for Systems Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/setup-instance-profile.html).

**Support for compressed playbook structure**

The AWS-ApplyAnsiblePlaybooks document allows you to run compressed .zip files in the downloaded bundle. The document checks if the downloaded files contain a compressed file in .zip format. If a .zip is found, the document automatically decompresses the file and then runs the specified Ansible automation.

**Enhanced logging**

The AWS-ApplyAnsiblePlaybooks document includes an optional parameter for specifying different levels of logging. Specify -v for low verbosity, -vv or –vvv for medium verbosity, and -vvvv for debug level logging. These options directly map to Ansible verbosity options.

**Ability to specify which playbook to run when playbooks are bundled**

The AWS-ApplyAnsiblePlaybooks document includes a required parameter for specifying which playbook to run when multiple playbooks are bundled. This option provides flexibility for running playbooks to support different use cases.

**9.Installed dependencies**

If you specify **True** for the **InstallDependencies** parameter, then Systems Manager verifies that your instances have the following dependencies installed:

* **Ubuntu Server/Debian Server**: Apt-get (Package Management), Python 3, Ansible, Unzip
* **Amazon Linux**: Ansible
* **RHEL**: Python 3, Ansible, Unzip

If one or more of these dependencies aren't found, then Systems Manager automatically installs them.

**10.Create an association that runs Ansible playbooks (console)**

The following procedure describes how to use the Systems Manager console to create a State Manager association that runs Ansible playbooks by using the AWS-ApplyAnsiblePlaybooks document.

**To create an association that runs Ansible playbooks (console)**

1. Open the AWS Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **State Manager**.

-or-

If the AWS Systems Manager home page opens first, choose the menu icon (  ) to open the navigation pane, and then choose **State Manager**.

1. Choose **State Manager**, and then choose **Create association**.
2. For **Name**, specify a name that helps you remember the purpose of the association.
3. In the **Document** list, choose **AWS-ApplyAnsiblePlaybooks**.
4. In the **Parameters** section, for **Source Type**, choose either **GitHub** or **S3**.

**GitHub**

If you choose **GitHub**, enter repository information in the following format.

{

"owner":"*user\_name*",

"repository":"*name*",

"path":"*path\_to\_directory\_or\_playbook\_to\_download*",

"getOptions":"*branch:branch\_name*",

"tokenInfo":"{{*(Optional)\_token\_information*}}"

}

**S3**

If you choose **S3**, enter path information in the following format.

{

"path":"https://s3.amazonaws.com/*path\_to\_directory\_or\_playbook\_to\_download*"

}

1. For **Install Dependencies**, choose an option.
2. (Optional) For **Playbook File**, enter a file name. If a Zip file contains the playbook, specify a relative path to the Zip file.
3. (Optional) For **Extra Variables**, enter variables that you want State Manager to send to Ansible at runtime.
4. (Optional) For **Check**, choose an option.
5. (Optional) For **Verbose**, choose an option.
6. For **Targets**, choose an option. For information about using targets, see [About targets and rate controls in State Manager associations](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-targets-and-rate-controls.html).
7. In the **Specify schedule** section, choose either **On schedule** or **No schedule**. If you choose **On schedule**, then use the buttons provided to create a cron or rate schedule for the association.
8. In the **Advanced options** section, for **Compliance severity**, choose a severity level for the association. Compliance reporting indicates whether the association state is compliant or noncompliant, along with the severity level you indicate here. For more information, see [About State Manager association compliance](https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-compliance-about.html#sysman-compliance-about-association).
9. In the **Rate control** section, configure options to run State Manager associations across a fleet of managed instances. For information about using rate controls, see [About targets and rate controls in State Manager associations](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-targets-and-rate-controls.html).

In the **Concurrency** section, choose an option:

* + Choose **targets** to enter an absolute number of targets that can run the association simultaneously.
  + Choose **percentage** to enter a percentage of the target set that can run the association simultaneously.

In the **Error threshold** section, choose an option:

* + Choose **errors** to enter an absolute number of errors that are allowed before State Manager stops running associations on additional targets.
  + Choose **percentage** to enter a percentage of errors that are allowed before State Manager stops running associations on additional targets.

1. (Optional) For **Output options**, to save the command output to a file, select the **Enable writing output to S3** box. Enter the bucket and prefix (folder) names in the boxes.

**Note**

The S3 permissions that grant the ability to write the data to an S3 bucket are those of the instance profile assigned to the managed node, not those of the IAM user performing this task. For more information, see [Create an IAM instance profile for Systems Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/setup-instance-profile.html) or [Create an IAM service role for a hybrid environment](https://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-service-role.html). In addition, if the specified S3 bucket is in a different AWS account, verify that the instance profile or IAM service role associated with the managed node has the necessary permissions to write to that bucket.

1. Choose **Create Association**.

**Note**

If you use tags to create an association on one or more target instances, and then you remove the tags from an instance, that instance no longer runs the association. The instance is disassociated from the State Manager document.

**11.Create an association that runs Ansible playbooks (CLI)**

The following procedure describes how to use the AWS Command Line Interface (AWS CLI) to create a State Manager association that runs Ansible playbooks by using the AWS-ApplyAnsiblePlaybooks document.

**To create an association that runs Ansible playbooks (CLI)**

1. Install and configure the AWS Command Line Interface (AWS CLI), if you haven't already.

For information, see [Install or upgrade AWS command line tools](https://docs.aws.amazon.com/systems-manager/latest/userguide/getting-started-cli.html).

1. Run one of the following commands to create an association that runs Ansible playbooks by targeting instances using Amazon Elastic Compute Cloud (Amazon EC2) tags. Command (A) specifies GitHub as the source type. Command (B) specifies Amazon S3 as the source type.

**(A) GitHub source**

* + [Linux & macOS](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-ansible.html#linux-&-macos)
  + [Windows](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-ansible.html#windows)

aws ssm create-association --name "AWS-ApplyAnsiblePlaybooks" \

--targets Key=tag:*TagKey*,Values=*TagValue* \

--parameters '{"SourceType":["GitHub"],"SourceInfo":["{\"owner\":\"*owner\_name*\", \"repository\": \"*name*\", \"getOptions\": \"branch:master\"}"],"InstallDependencies":["*True\_or\_False*"],"PlaybookFile":["*file\_name*.yml"],"ExtraVariables":["*key/value\_pairs\_separated\_by\_a\_space*"],"Check":["*True\_or\_False*"],"Verbose":["*-v,-vv,-vvv, or -vvvv*"]}' \

--association-name "*name*" --schedule-expression "*cron\_or\_rate\_expression*"

Here is an example.

aws ssm create-association --name "AWS-ApplyAnsiblePlaybooks" \

--targets "Key=tag:OS,Values=Linux" \

--parameters '{"SourceType":["GitHub"],"SourceInfo":["{\"owner\":\"ansibleDocumentTest\", \"repository\": \"Ansible\", \"getOptions\": \"branch:master\"}"],"InstallDependencies":["True"],"PlaybookFile":["hello-world-playbook.yml"],"ExtraVariables":["SSM=True"],"Check":["False"],"Verbose":["-v"]}' \

--association-name "AnsibleAssociation" --schedule-expression "cron(0 2 ? \* SUN \*)"

**(B) S3 source**

* + [Linux & macOS](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-ansible.html#linux-&-macos)
  + [Windows](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-ansible.html#windows)

aws ssm create-association --name "AWS-ApplyAnsiblePlaybooks" \

--targets Key=tag:*TagKey*,Values=*TagValue* \

--parameters '{"SourceType":["S3"],"SourceInfo":["{\"path\":\"https://s3.amazonaws.com/*path\_to\_Zip\_file,\_directory,\_or\_playbook\_to\_download*\"}"],"InstallDependencies":["*True\_or\_False*"],"PlaybookFile":["*file\_name*.yml"],"ExtraVariables":["*key/value\_pairs\_separated\_by\_a\_space*"],"Check":["*True\_or\_False*"],"Verbose":["*-v,-vv,-vvv, or -vvvv*"]}' \

--association-name "*name*" --schedule-expression "*cron\_or\_rate\_expression*"

Here is an example.

aws ssm create-association --name "AWS-ApplyAnsiblePlaybooks" \

--targets "Key=tag:OS,Values=Linux" \

--parameters '{"SourceType":["S3"],"SourceInfo":["{\"path\":\"https://s3.amazonaws.com/DOC-EXAMPLE-BUCKET/playbook.yml\"}"],"InstallDependencies":["True"],"PlaybookFile":["playbook.yml"],"ExtraVariables":["SSM=True"],"Check":["False"],"Verbose":["-v"]}' \

--association-name "AnsibleAssociation" --schedule-expression "cron(0 2 ? \* SUN \*)"

**Note**

State Manager associations don't support all cron and rate expressions. For more information about creating cron and rate expressions for associations, see [Reference: Cron and rate expressions for Systems Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/reference-cron-and-rate-expressions.html).

The system attempts to create the association on the instances and immediately apply the state.

1. Run the following command to view an updated status of the association you just created.

aws ssm describe-association --association-id "*ID*"

**12.Walkthrough: Using Automation with Jenkins**

If your organization uses Jenkins software in a CI/CD pipeline, you can add Automation as a post-build step to pre-install application releases into Amazon Machine Images (AMIs). Automation is a capability of AWS Systems Manager. You can also use the Jenkins scheduling feature to call Automation and create your own operating system (OS) patching cadence.

The example below shows how to invoke Automation from a Jenkins server that is running either on-premises or in Amazon Elastic Compute Cloud (Amazon EC2). For authentication, the Jenkins server uses AWS credentials based on an AWS Identity and Access Management (IAM) user that you create in the example. If your Jenkins server is running in Amazon EC2, you can also authenticate it using an IAM instance profile role.

**Note**

Be sure to follow Jenkins security best practices when configuring your instance.

**Before you begin**

Complete the following tasks before you configure Automation with Jenkins:

* Complete the [Walkthrough: Simplify AMI patching using Automation, AWS Lambda, and Parameter Store](https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-walk-patch-windows-ami-simplify.html) example. The following example uses the **UpdateMyLatestWindowsAmi** runbook created in that example.
* Configure IAM roles for Automation. Systems Manager requires an instance profile role and a service role ARN to process automations. For more information, see [Setting up Automation](https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-setup.html).
* After you configure IAM roles for Automation, use the following procedure to create an IAM user account for your Jenkins server. The automation uses the IAM user account's Access key and Secret key to authenticate the Jenkins server during the automation.

**To create a user account for the Jenkins server**

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. Choose the **JSON** tab.
4. Replace the default content with the following. Be sure to replace *us-west-2* and *123456789012* with the Region and account you want to use.
5. {
6. "Version": "2012-10-17",
7. "Statement": [
8. {
9. "Effect": "Allow",
10. "Action": "ssm:StartAutomationExecution",
11. "Resource": [
12. "arn:aws:ssm:*us-west-2*:*123456789012*:document/UpdateMyLatestWindowsAmi",
13. "arn:aws:ssm:*us-west-2*:*123456789012*:automation-definition/UpdateMyLatestWindowsAmi:$DEFAULT"
14. ]
15. }
16. ]

}

1. Choose **Review policy**.
2. On the **Review policy** page, for **Name**, enter a name for the inline policy, such as **JenkinsPolicy**.
3. Choose **Create policy**.
4. In the navigation pane, choose **Users**.
5. Choose **Add user**.
6. In the **Set user details** section, specify a user name (for example, Jenkins).
7. In the **Select AWS access type** section, choose **Programmatic Access**.
8. Choose **Next:Permissions**.
9. In the **Set permissions for** section, choose **Attach existing policies directly**.
10. In the filter field, enter the name of the policy you created earlier.
11. Select the check box next to the policy, and then choose **Next: Tags**.
12. (Optional) Add one or more tag key-value pairs to organize, track, or control access for this user, and then choose **Next: Review**.
13. Verify the details, and then choose **Create**.
14. Copy the access and secret keys to a text file. You will specify these credentials in the next procedure.

Use the following procedure to configure the AWS CLI on your Jenkins server.

**To configure the Jenkins server for Automation**

1. Connect to your Jenkins server on port 8080 using your preferred browser to access the management interface.
2. Enter the password found in /var/lib/jenkins/secrets/initialAdminPassword. To display your password, run the following command.

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

1. The Jenkins installation script directs you to the **Customize Jenkins** page. Select **Install suggested plugins**.
2. Once the installation is complete, choose **Administrator Credentials**, select **Save Credentials**, and then select **Start Using Jenkins**.
3. In the left navigation pane, choose **Manage Jenkins**, and then choose **Manage Plugins**.
4. Choose the **Available** tab, and then enter **Amazon EC2 plugin**.
5. Select the check box for **Amazon EC2 plugin**, and then select **Install without restart**.
6. When the installation completes, select **Go back to the top page**.
7. Choose **Manage Jenkins**, and then choose **Configure System**.
8. In the **Cloud** section, select **Add a new cloud**, and then choose **Amazon EC2**.
9. Enter your information in the remaining fields. You must enter your AWS credentials in the **Add Credentials** field.

Use the following procedure to configure your Jenkins project to invoke Automation.

**To configure your Jenkins server to invoke Automation**

1. Open the Jenkins console in a web browser.
2. Choose the project that you want to configure with Automation, and then choose **Configure**.
3. On the **Build** tab, choose **Add Build Step**.
4. Choose **Execute shell** or **Execute Windows batch command** (depending on your operating system).
5. In the **Command** box, run an AWS CLI command like the following.

aws --region *the AWS Region of your source AMI* ssm start-automation-execution --document-name *your runbook name* --parameters *parameters for the runbook*

The following example command uses the **UpdateMyLatestWindowsAmi** runbook and the Systems Manager Parameter latestAmi created in [Walkthrough: Simplify AMI patching using Automation, AWS Lambda, and Parameter Store](https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-walk-patch-windows-ami-simplify.html).

aws --region *region-id* ssm start-automation-execution \

--document-name UpdateMyLatestWindowsAmi \

--parameters \

"sourceAMIid='{{ssm:latestAmi}}'"

In Jenkins, the command looks like the example in the following screenshot.


                            Jenkins information
                        

1. In the Jenkins project, choose **Build Now**. Jenkins returns output similar to the following example.


                            Jenkins information
                        