MODULE 2 · STUDY NOTES

Tooling and Automation

AWS Systems Manager's automation suite, the CLI / PowerShell / SDK toolset, infrastructure as code with CloudFormation and OpsWorks, and hosting a static website on Amazon S3

Operating a fleet of servers by hand — logging into each one, running the same command, patching them one at a time — does not scale, and it is exactly what this module replaces with **tooling and automation**. The centerpiece is **AWS Systems Manager**, a single management service whose nine features let you inventory, patch, connect to, configure, and audit EC2 and on-premises instances from one place, with automation woven through all of it. The module then broadens out to the wider AWS toolset: the **AWS Command Line Interface (AWS CLI)**, **AWS Tools for PowerShell**, and language **SDKs/APIs** that let you script and manage infrastructure as code, and the **CloudFormation** and **OpsWorks** services that provision and configure whole environments predictably and repeatedly. Finally it puts one tool to practical work — using **Amazon S3** to host a static website — and closes with a test-taking strategy for choosing the right tool from a scenario. Two ideas recur: automate the repetitive, and pick the tool the problem actually calls for.

LEARNING OBJECTIVES

- Describe the purpose and function of AWS Systems Manager and its related features
- Describe the purpose and function of AWS Tools for PowerShell
- Identify additional development tools used for tooling and automation, such as SDKs, AWS CloudFormation, and AWS OpsWorks
- Explain how Amazon S3 can be used to host a static website

2.1 AWS Systems Manager

AWS Systems Manager is a management service that helps a systems operations specialist collect software inventory, apply operating system patches, create system images, and configure Windows and Linux operating systems. It is designed to be **highly automation-focused**, enabling the configuration and management of systems running **on-premises or in AWS**. The workflow is simple to describe: you select the instances you want to manage — individually or by tag — and define the management tasks you want to perform. Everything else in this section is one of the features Systems Manager provides to carry those tasks out.

WHAT SYSTEMS MANAGER HELPS YOU DO

- Collect software **inventory** about instances and the software installed on them
- Apply **operating system patches** automatically and on a schedule
- Create **system images**
- Configure **Windows and Linux** operating systems
- Manage instances running **on-premises or in AWS** from a single place

TABLE 2.1 AWS Systems Manager features at a glance

FEATURE	WHAT IT DOES	REMEMBER
Automation	Runs common, repetitive IT tasks as a series of steps defined in an SSM document, across a collection of resources	Custom automations in JSON; CloudWatch Events can trigger it
Run Command	Runs predefined or custom commands against instances chosen by instance or tag	No bastion host or SSH keys; IAM permissions + CloudTrail audit
Session Manager	Opens an interactive, browser-based shell to an instance	No open inbound ports; logs to CloudWatch Logs or S3
Patch Manager	Automatically deploys OS and software patches across large instance groups	Patch baseline + maintenance window + reboot
Maintenance Windows	Schedules recurring administrative and maintenance tasks	Concurrency and error-rate limits; runs Run Command, Automation, Step Functions, Lambda
State Manager	Keeps EC2 and hybrid instances in a defined configuration state	SSM doc on a schedule; output can go to S3
Parameter Store	Stores configuration data and secrets for apps and scripts	IAM-controlled; KMS-encryptable
Inventory	Collects configuration and installed-software data across instances	No need to log into each instance
Insights	Dashboard of operational data for each resource group	Aggregates CloudTrail, Config, inventory, patch data

Automation and Run Command handle remote execution. The **Automation** feature defines common IT tasks as a collection of steps in an **AWS Systems Manager document (SSM document)** and then runs all of those steps across an entire collection of AWS resources — for example, to remediate unreachable instances, create golden **Amazon Machine Images (AMIs)**, or patch instances. Custom automations can be authored in JSON, and **Amazon CloudWatch Events** can be configured to trigger them. A typical development approach is to create an automation document (or start from an existing template) with sequential steps and parameters, run it (launch an instance, take a snapshot, tag instances, delete old images, terminate an instance), and then monitor the workflow and confirm the expected results. **Run Command** provides a simpler path for one-off work: it runs predefined commands — or ones you create — against EC2 instances, reducing management overhead because you manage instances **without setting up bastion hosts or managing SSH keys and certificates**. Through integration with IAM you apply granular permissions over what users can do, and **AWS CloudTrail** records every action so changes can be audited.

Session Manager lets you manage an EC2 instance through an **interactive, browser-based shell** available in the AWS Management Console (the underlying command is `aws start-session --target`). Like Run Command, it provides secure and auditable instance management **without opening inbound ports in security groups, maintaining bastion hosts, or managing SSH keys**, and it makes it straightforward to comply with corporate policies that require controlled access, strict security practices, and **auditable logs** — session activity can be logged to **Amazon CloudWatch Logs** or an **Amazon S3 bucket**. It works for both Windows and Linux instances.

Patch Manager and Maintenance Windows automate patching. Patching servers is time-consuming, repetitive, error-prone (mistakes cause downtime), and tied to compliance requirements — good reasons to automate it. **Patch Manager** deploys OS and software patches automatically across large groups of EC2 or on-premises instances: you create a **patch**

baseline (rules that automatically approve or reject patches), define a **maintenance window** and group instances for patching, apply patches during the window and reboot every instance in the patch group, then review the results and patch-compliance details. **Maintenance Windows** is the scheduling feature underneath: it lets you schedule regular tasks to run automatically, set limits on the simultaneous running of tasks and on allowable error rates, and assign tasks to targets. Those tasks can be **Run Command** commands, **Automation** workflows, **AWS Step Functions** workflows, or **AWS Lambda** functions.

State Manager is a secure, scalable configuration management feature that automates keeping your EC2 and hybrid infrastructure in a state that you define. It can install software when instances are created, download and update agents, configure network settings, join a Windows domain, patch instances, and run scripts on Linux and Windows managed instances. You create or identify an **SSM document** — authored in **JSON or YAML** with the steps and parameters you specify — then associate your managed instances with it, defining a **schedule** for how often to apply the configured state; you can optionally write the command output to an **Amazon S3 bucket**.

Parameter Store replaces sensitive values kept in configuration files or hard-coded into source code. You store configuration information — **passwords, keys, license codes, and database strings** — as parameters and reference them from applications or scripts. Parameter Store integrates with **IAM** to control access to each parameter, and stored information can be encrypted with **AWS Key Management Service (AWS KMS)**. One limitation to know: **SSM documents currently do not support references to SecureString parameters**, so to use a SecureString with Run Command you must retrieve the parameter value first and then pass it in.

Inventory and Insights give you fleet-wide visibility. The **Inventory** feature collects information about instances and the software installed on them — application data, network configurations, server roles, files, updates, Windows services, and system properties — giving a comprehensive picture of configurations and installed applications across many instances **without logging into each one**; the gathered data supports managing application assets, tracking licenses, monitoring file integrity, and discovering applications not installed by a traditional installer. The **Insights** dashboard aggregates and displays operational data for each resource group in one place, eliminating the need to navigate across multiple consoles. It surfaces **API calls from CloudTrail, resource configuration changes from AWS Config, software inventory, and patch-compliance status by resource group**, and you can integrate CloudWatch dashboards, Trusted Advisor notifications, and Personal Health Dashboard alerts into it.

WORKED EXAMPLE – THE SYSTEMS MANAGER LAB (FOUR FEATURES)

The module's hands-on lab exercises four features against a managed EC2 instance. You use **Inventory** to verify configurations and permissions; **Run Command** to run the `AWS-RunShellScript` command that installs Apache HTTP Server, PHP, the AWS SDK for PHP, and the Widget Manufacturing application; **Parameter Store** to store a configuration value that activates a feature in that application; and **Session Manager** to open a browser command session to the instance and enter shell commands directly — all without SSH keys or a bastion host.

COMMON PITFALL

A Parameter Store **SecureString** cannot be referenced directly inside an SSM document. Because SSM documents do not support SecureString references, you must retrieve the decrypted value first and then pass it to the feature that needs it (such as Run Command). Plain String and StringList parameters do not have this restriction.

SECTION TAKEAWAY

- AWS Systems Manager lets you **safely automate common and repetitive IT operations** and management tasks across AWS resources.
- It provides a **suite of features** that automate operational tasks across both AWS and on-premises resources.
- **Patch Manager and Maintenance Windows** together can apply operating system patches on a **predefined schedule**.

2.2 Additional administration and development tools

Beyond Systems Manager, AWS offers a range of tools for administering resources and building automation. **AWS Tools for PowerShell** enable you to script operations on your AWS resources from the **PowerShell command line** — they let you perform many of the same actions available in the AWS SDK for .NET. They are useful for quick tasks such as creating and configuring security groups, launching instances, creating and deleting S3 buckets, performing IAM tasks (like assigning a role to a user), and publishing CloudWatch metrics. The AWS Tools for Windows PowerShell provide a set of **cmdlets** — commands used in a Windows PowerShell environment to perform an action — and support the **same set of services and Regions** as the AWS SDKs.

WORKED EXAMPLE – POWERSHELL CMDLETS FOLLOW A VERB-NOUN PATTERN

The module walks through several cmdlets that show the consistent `Verb-Noun` naming. `New-EC2Instance -ImageId ami-c49c0dac -MinCount 1 -MaxCount 1 -InstanceType t1.micro` launches an EC2 instance; `Get-EC2Instance -Filter $filter_reservation` views instance details filtered on a reservation ID; `New-EC2SecurityGroup` and `Get-EC2SecurityGroup` create and view a security group in a VPC; `New-S3Bucket -BucketName website-example -Region us-west-1` creates a bucket; and `New-IAMGroup`, `New-IAMUser`, and `Add-IAMUserToGroup` create an IAM group and user and add the user to the group (the `-Path` argument is optional). One consistent verb set, applied to different AWS nouns.

SDKs, APIs, and infrastructure as code. The concept of **infrastructure as code** is fundamental to cloud computing and differentiates the cloud from more traditional IT environments. The AWS **software development kits (SDKs)** and **application programming interfaces (APIs)** provide the tools needed for applications built on AWS to be managed as infrastructure as code. AWS provides **language-specific SDKs** — with APIs — for **JavaScript, Python, PHP, .NET, Ruby, Go, Node.js, C++, and Java**. Both the SDKs and APIs let a developer incorporate the connectivity and functionality of the wide range of AWS services into their code **without writing functions from scratch**, and each is backed by extensive documentation, getting-started and developer guides, API references, and community forums.

AWS CloudFormation lets you create, update, and delete a set of resources spanning multiple AWS services as a **single unit called a stack**. An entire infrastructure is modeled in a single text file — a **template written in JSON or YAML** — that defines all the desired resources; a **stack** is that collection of resources, managed as one unit, so you create, update, or delete the resources by creating, updating, or deleting the stack. Resources in a stack can include EC2 instances, Amazon RDS instances, VPCs, and many others. Three features stand out: you can **preview how proposed changes will impact the environment** before running them (CloudFormation only makes changes after you decide to run them), you can **detect drift** — a drift-detection operation determines whether a stack has diverged from its expected template configuration — and you can **enable extensibility** with custom logic built using **AWS Lambda** (for example, a function that looks up the most recent AMI IDs to use in a stack).

AWS OpsWorks is a **configuration management service** that provides managed instances of **Chef and Puppet** — automation platforms that let you use code to automate the configuration of your servers. OpsWorks enables **infrastructure as code** and automates **infrastructure, application management, compliance testing, and**

configuration tasks across your EC2 instances or on-premises compute environments. It comes in **three offerings**: AWS OpsWorks for Chef Automate, AWS OpsWorks for Puppet Enterprise, and AWS OpsWorks Stacks.

TABLE 2.2 Comparing the administration and development tools

TOOL	WHAT IT IS	WRITTEN IN / BASED ON	TYPICAL USE
AWS Tools for PowerShell	Cmdlets for the PowerShell command line	PowerShell cmdlets	Quick scripted admin tasks; same actions as the SDK for .NET
SDKs & APIs	Language-specific developer kits	Java, Python, .NET, JS, Node.js, Ruby, PHP, Go, C++	Build AWS functionality into application code (IaC)
AWS CloudFormation	Provisions a stack of resources across services	JSON or YAML templates	Repeatable infrastructure; preview, drift detection, Lambda extensibility
AWS OpsWorks	Configuration management service	Managed Chef and Puppet	Automate server configuration, deployment, and management

COMMON PITFALL

CloudFormation templates are ordinary text files in **JSON or YAML** — the same formats used for SSM and State Manager documents — not a proprietary AWS language. And CloudFormation never changes running resources until you choose to run the change; you can review proposed changes and run drift detection first.

SECTION TAKEAWAY

- **AWS Tools for PowerShell** supports scripting operations on AWS resources from the PowerShell command line.
- **AWS CloudFormation** enables you to create, update, and delete AWS infrastructure deployments **predictably and repeatedly**.
- **AWS OpsWorks** is a configuration management service (managed Chef and Puppet) that automates infrastructure, application management, compliance testing, and configuration tasks.

2.3 Hosting a static website on Amazon S3

Amazon S3 provides object storage, and one practical use of it is to **host a static website**. Doing so avoids deploying complex or costly runtime infrastructure. A **static website** consists of individual webpages that include static content and optional **client-side scripts** — the pages do **not** rely on any server-side processing, so there is no application server to run. Hosting one on S3 takes a few steps, after which clients reach the site at the endpoint URL that S3 assigns.

STEPS TO HOST A STATIC WEBSITE ON AMAZON S3

- **Create a bucket** in Amazon S3 to store the website content.
- **Configure the bucket** to enable website hosting and grant **public read permission** to its content.
- **Upload the website content** to the bucket using the AWS Management Console or the AWS CLI.
- **Access the website** at the endpoint URL Amazon S3 assigns — it includes the bucket name and the Region name.

The **endpoint URL** conforms to one of two formats, and the difference matters: the separator character **before the Region name is either a dash or a period, depending on the Region** that contains the bucket. A bucket in US West (Oregon) uses a dash; a bucket in EU (Frankfurt) uses a period. Two additional requirements: uploaded content should be stored in a **folder hierarchy that reflects the website's content structure**, and when you enable a bucket for website hosting you must provide the name of an **index document** — the default webpage S3 returns when a request is made to the root of the site or one of its subfolders. The default index document name is `index.html`, and the loaded content must include it at the proper folder level.

TABLE 2.3 S3 static-website endpoint URL formats

ENDPOINT URL FORMAT	SEPARATOR	EXAMPLE REGION
http://bucket-name.s3-website-region.amazonaws.com	Dash (-)	US West (Oregon), us-west-2
http://bucket-name.s3-website.region.amazonaws.com	Period (.)	EU (Frankfurt), eu-central-1

Using a custom domain. Instead of the raw S3 endpoint URL, you can use **Amazon Route 53** to map your own domain name to the S3 endpoint — for example a root domain like mompopcafe.com, or a subdomain like www.mompopcafe.com. To support both, you create and configure **two buckets**, one for each name: one bucket **contains the website content**, and the other **redirects** requests to the content bucket. The **bucket names must match the domain names** they serve, and you create **alias records** in Route 53 that map the domain and subdomain to the corresponding S3 bucket endpoints. The four steps to set up a static website with a custom domain are: **register a domain**, **create and configure a bucket and upload data**, **add alias records**, and **test**.

> **WORKED EXAMPLE – THE CAFÉ WEBSITE ACTIVITY USES THE AWS CLI**
>
> In the module's activity, a static website for the Mom & Pop Café & Bakery is deployed to Amazon S3 using the **AWS CLI**: create an S3 bucket, create an IAM user with full access to S3, upload the HTML files and images, and create a **batch file** that re-uploads the site whenever local files change. Using the CLI rather than the Console lets **multiple images upload at one time and work programmatically** — a real advantage once the owner starts adding new menu photos frequently.

> **COMMON PITFALL**
>
> A static website on S3 needs **no web or application server** — there is no server-side processing — but it is not zero-configuration. You must **enable website hosting**, **grant public read permission**, and **define an index document** (default `index.html`) at the correct folder level, or requests to the site root will fail.

> **SECTION TAKEAWAY**
>
> - **Amazon S3 provides object storage**, and one of its uses is hosting a static website.
> - Hosting on S3 helps users **avoid complex or costly runtime infrastructures**.
> - A custom domain adds **Amazon Route 53** alias records and a redirect bucket on top of the content bucket.

2.4 Choosing the right tool – test strategies

The module closes with a **test-taking strategy** rather than a new service: when a scenario question asks which tool to use, read the scenario and the actual question **before** looking at the answer choices, then isolate the **keywords** and use them to rule out distractors and narrow to the correct answer. The recurring lesson of the module is to match the tool to

what the task really requires — the Console for visual, one-off work; the CLI, PowerShell, or SDKs for programmatic, repeatable, bulk operations.

HOW TO READ A SCENARIO QUESTION

- Fully understand the **scenario and the question** before reading the answer choices.
- Identify and highlight the **keywords** in the scenario and question.
- Use those keywords to **eliminate distractors** and narrow down the correct answer.

WORKED EXAMPLE – THE MODULE'S SCENARIO QUESTION

A financial company's administrators update website images monthly; the images load slowly for some global customers. An administrator must update this month's images while archiving last month's. Which tool expedites the request — AWS CLI, AWS Management Console, SDKs, all of the above, or none of the above? Isolate the keywords: **update images**, **archive**, and **expedite**. The task is a bulk file operation, and the **AWS CLI** is the best fit — it can **upload and archive many objects at once and work programmatically** (even from a batch file), which the Console's one-file-at-a-time workflow cannot expedite, and it needs no custom code the way an SDK would. The detail about slow loading for global customers is a **distractor** — that is a content-delivery concern, not part of the image-update task being asked about.

COMMON PITFALL

Do not answer from the flashiest keyword. A phrase like “slow for global customers” tempts a content-delivery answer, but the **question** was about updating and archiving images — match the tool to what is actually being asked, not to an unrelated symptom mentioned in the scenario.

Module summary

- AWS Systems Manager is one automation-focused management service for EC2 and on-premises instances; it helps you collect software inventory, apply OS patches, create system images, and configure Windows and Linux — you choose the instances and define the tasks.
- Its nine features cover Automation and Run Command (remote task/command execution), Session Manager (browser shell), Patch Manager and Maintenance Windows (scheduled patching), State Manager (defined configuration state), Parameter Store (config and secrets), Inventory, and the Insights dashboard.
- Run Command and Session Manager eliminate bastion hosts, open inbound ports, and SSH keys; IAM provides granular permissions and AWS CloudTrail records actions for audit — with Session Manager logging to CloudWatch Logs or S3.
- SSM, State Manager, and CloudFormation artifacts are plain-text documents/templates in JSON or YAML; Patch Manager relies on a patch baseline, and Maintenance Window tasks can be Run Command, Automation, Step Functions, or Lambda.
- AWS Tools for PowerShell script AWS operations with cmdlets (same actions as the SDK for .NET); language SDKs and APIs (Java, Python, .NET, JavaScript, Node.js, Ruby, PHP, Go, C++) let developers manage AWS as infrastructure as code.
- AWS CloudFormation creates, updates, and deletes a stack of resources across services from a JSON/YAML template — with change preview, drift detection, and Lambda extensibility — while AWS OpsWorks provides managed Chef and Puppet configuration management (three offerings).

- Amazon S3 can host a static website: create a bucket, enable website hosting and public read, upload files (Console or CLI), and define an index document (default index.html); the endpoint URL carries the bucket name and Region, and Amazon Route 53 alias records map a custom domain.
- Choose the tool the task actually needs — the Console for visual one-off work, the CLI/PowerShell/SDKs for programmatic, repeatable, bulk operations — and on scenario questions, read the keywords to rule out distractors.

Review questions

1. **What is AWS Systems Manager, and name four things it helps you do?**

   Answer: An automation-focused management service for AWS and on-premises instances. It helps you collect software inventory, apply operating system patches, create system images, and configure Windows and Linux operating systems — you select the instances (by ID or tag) and define the tasks.

2. **How do Run Command and Session Manager improve security over traditional instance access?**

   Answer: They manage instances without bastion hosts, open inbound ports, or SSH keys. IAM applies granular permissions, AWS CloudTrail records the actions for audit, and Session Manager logs session activity to CloudWatch Logs or an S3 bucket.

3. **Describe the Patch Manager workflow.**

   Answer: Create a patch baseline (rules that automatically approve or reject patches); define a maintenance window and group the instances; apply patches during the window and reboot every instance in the patch group; then review the results and patch-compliance details.

4. **Why use Parameter Store, and what is the SecureString caveat?**

   Answer: It stores configuration data and secrets — passwords, keys, license codes, database strings — instead of hard-coding them, with IAM access control and optional KMS encryption. Caveat: SSM documents cannot reference SecureString parameters directly, so you must retrieve the value first before passing it (for example, to Run Command).

5. **Compare AWS CloudFormation and AWS OpsWorks.**

   Answer: CloudFormation uses JSON/YAML templates to create, update, and delete a stack of resources spanning multiple services, with change preview, drift detection, and Lambda extensibility. OpsWorks is a configuration management service that provides managed Chef and Puppet (three offerings) to automate how servers are configured, deployed, and managed.

6. **What are the steps to host a static website on Amazon S3?**

   Answer: Create a bucket for the content; configure it to enable website hosting and grant public read permission; upload the files (via the Console or the AWS CLI) and define the index document (default index.html); then access the site at the S3 endpoint URL. Optionally use Amazon Route 53 alias records for a custom domain.

7. **Why is the AWS CLI the right tool to expedite bulk image updates on the website?**

   Answer: The CLI can upload and archive many objects at once and work programmatically — even from a batch file — which is faster than the Management Console's one-file-at-a-time workflow and needs no custom code like an SDK would.

8. **What is infrastructure as code, and which tools in this module enable it?**

   Answer: Managing and provisioning infrastructure through code and templates rather than manual configuration. It is enabled by the AWS SDKs and APIs, AWS Tools for PowerShell, AWS CloudFormation, and AWS OpsWorks.