MODULE 10 · STUDY NOTES

Managing Resource Consumption

Tagging resources for cost and control, enforcing tags with IAM and AWS Config, the AWS cost-management toolkit, and using Trusted Advisor to cut waste

Cloud makes it trivially easy to launch resources — and just as easy to lose track of what they are, who owns them, and what they cost. This module gives you the tools to keep resource consumption under control. It begins with **tagging**: the humble key-value label that turns an anonymous pile of instance IDs into an organized inventory you can search, secure, automate, and bill by department. It then shows how to make tagging stick — an **IAM policy** to require tags at creation, **AWS Config** to catch resources that slipped through, and scripts (from a simple stopinator to a serverless Lambda) that act on those tags to shut down idle capacity. Next come the **cost-management tools** — AWS Budgets, Cost Explorer, the Cost and Usage Report, CloudWatch billing alarms, and the Billing Dashboard — plus the design principles (right-sizing, Reserved and Spot Instances, serverless, managed services) that reduce spend in the first place. Finally, **AWS Trusted Advisor** ties it together, scanning your account against AWS best practices and pointing you straight at the waste.

LEARNING OBJECTIVES

- Explain the purpose and function of tagging in AWS
- Describe the cost management strategies associated with tagging
- Describe how to enforce tagging by using AWS Identity and Access Management (IAM) policies
- Identify some of the cost benefits of the cloud
- Explain the purpose and function of the AWS Trusted Advisor service
- Manage resources with tagging

10.1 Tagging fundamentals

A **tag** is a label you assign to an AWS resource so that you can identify or categorize it in a meaningful way. Each tag is a **key-value pair** that you define. Consider two Amazon EC2 instances in a development environment: give both a tag with the key `Name`, valued `CommandHost` on the first and `DatabaseServer` on the second, and their purpose is obvious at a glance — far easier than reading instance IDs like `i-123` and `i-456`. Add a second tag with the key `Environment`, valued `Development` or `Test`, and you can also tell which environment each instance runs in. Without tags, the same instances are just anonymous IDs scattered across VPCs; with tags, their purpose and environment are self-evident.

TAG CHARACTERISTICS YOU MUST KNOW

- **Assigned after creation** — you can only assign a tag to a resource once it exists (though the CLI/API can attach tags to EC2 instances and EBS volumes as part of the call that creates them).
- **Case-sensitive** — keys and values are case-sensitive; a best practice is to use a consistent, standardized format so tags group correctly.
- **`aws:` prefix tags are reserved** — you cannot edit or delete tags with the `aws:` prefix. AWS generates them, for example `aws:createdBy` for cost allocation (identifies the creator) and the `aws:cloudformation:` stack tag that CloudFormation adds to every stack.
- **Sometimes inherited/propagated** — services that create other resources (Auto Scaling, AWS CloudFormation, Elastic Beanstalk) generally tag those resources with a reference to themselves; a resource made by a CloudFormation stack inherits the stack's tag.
- **Up to 50 tags per resource** — the limit is 50 user tags; tags with the `aws:` prefix do not count against it.

Good tags represent **organizationally relevant dimensions** — the axes your business actually manages resources along. Choosing them well is what makes tags useful for resource inventory, access control, cost tracking, automation, and general organization.

KEY TERMS

Environment
Distinguishes stages such as production or test

Application
Identifies the application a resource supports

Owner
Who is responsible for the resource

Department / Cost center
Ties the resource to an internal group for chargeback and cost allocation

Purpose
Why the resource exists

Stack
The deployment stack a resource belongs to

10.2 Enforcing and automating tags

Tags are only valuable if they are applied consistently, so AWS gives you both a **detective** control and a **preventive** control. The detective control is **AWS Config**, which provides predefined, customizable **managed rules** that evaluate whether your resources comply with best practices. The **required-tags** managed rule quickly assesses whether a specific tag is applied to your resources: you specify the required tag key and, optionally, its value (separate multiple values with commas). After you activate the rule, AWS Config compares your resources to the defined conditions and reports any non-compliant resource — evaluated when a resource changes or on a periodic basis.

The preventive control is an **IAM policy**, which enforces the use of specific tags through the **Condition** policy element. In the module's example, a request to create an EBS volume is allowed only when it carries exactly a `costcenter` tag valued `115` and a `department` tag valued `Accounting`. The values are pinned with `aws:RequestTag/costcenter` and `aws:RequestTag/department`, and `ForAllValues:StringEquals` on `aws:TagKeys` restricts the request to only those

two tags. The net effect: every newly created EBS volume must carry a `costcenter` and a `department` tag with the required values. Other tag rules you can enforce this way include blocking deletion of tags required by corporate standards, disallowing new tags on specific existing resources, and requiring encryption for any EBS volume created with a particular tag value.

TAGGING AND QUERYING FROM THE AWS CLI

Create a tag and then find resources by it. To tag an instance with a timestamp: `aws ec2 create-tags --resources i-1234567890abcdef0 --tags Key=SecurityCheck,Value=$TIMESTAMP`. The `--tags` option supplies the key and value. To list every instance in the current Region that carries that tag: run `aws ec2 describe-instances` with `--filters Name=tag-key,Values=SecurityCheck` (which key to filter on) and a `--query` expression such as `Reservations[].Instances[].[InstanceId,Tags[?Key=='SecurityCheck'].Value]` (which data to return).

COMMON USE CASES FOR TAGGING

- **Scheduled start/stop** — simultaneously shut down or restart all instances that carry a specific tag; for example, shut down every `Development` instance on weekends and restart it Monday to save cost.
- **Tag or terminate** — a script periodically checks that required tags exist and terminates non-compliant instances (the Netflix **Conformity Monkey** pattern). In practice it is staggered: Phase 1 emails the IAM user who created the instance to warn them, Phase 2 actually shuts the instance down.
- **Cost allocation and billing** — once resources are tagged by role and function, organize billing reports around those tags for more accurate cost allocation that reflects internal cost structures.

TAGGING BEST PRACTICES

- Use **automated tools** to manage tags rather than editing them by hand.
- The **Resource Groups Tagging API** enables programmatic control: tag and untag supported resources, use tag-based filters to search for resources, list all existing tag keys, and list all values for a specified key.
- Define tags along dimensions that facilitate **access control, cost tracking, automation, and organization**.

COMMON PITFALL

Do not confuse the two controls. **AWS Config's required-tags rule is detective** — it finds and reports resources that are already missing tags. **An IAM policy is preventive** — its Condition element stops a non-compliant resource from ever being created. A robust tagging strategy usually uses both: IAM to mandate tags on creation, Config to periodically verify that everything remains tagged.

10.3 Cost management tools and best practices

A fundamental benefit of the cloud is that you **pay only for what you need, when you need it** — which motivates you to use resources efficiently and shut off what you are not using. Practical strategies include creating **scripts or templates** that start up and shut down entire environments, and turning off unused resources: development or test environments at project end, specific services after business hours and during holidays, disaster-recovery environments until they are actually needed, and temporary instances (tagged as such) after their time period ends.

AWS also provides tools and services to access, organize, understand, control, and optimize your costs and usage. You can view overall month-to-date status in the **AWS Billing Dashboard**, whose **Spend Summary** shows last month's

spend, month-to-date spend, and a forecast for the month, while the **Month-to-Date Spend by Service** graph shows which services drive your costs. The four core cost tools below go deeper.

TABLE 10.1 AWS cost management tools and services

TOOL	WHAT IT DOES	KEY FACTS
AWS Budgets	Set custom cost and usage budgets; alerts you when thresholds are exceeded or forecast to be exceeded	Track by month, quarter, or year; custom start/end dates; notify via email and/or Amazon SNS
AWS Cost Explorer	View and analyze costs and usage to identify trends, cost drivers, and anomalies; build custom reports	Filter/group by service, instance type, tag; 13 months of history; 3-month forecast; RI recommendations; export to .csv
Cost and Usage Report	Generates a report with the most granular data about your costs and usage	Best for detailed, line-item analysis of what underlies your costs
CloudWatch billing alarm	Monitors estimated charges and alerts when they exceed a threshold you specify	Enable in the console; must be created in us-east-1; based on total + per-service metrics; sends SNS email

A few nuances matter on the exam. **Cost Explorer** can also view costs across all linked accounts when you have enabled **consolidated billing for AWS Organizations**, and it offers Reserved Instance utilization and coverage reports. A **CloudWatch billing alarm** stores its billing metric data in **us-east-1** (the central store for all billing metrics, representing worldwide charges); it triggers only when **actual** billing exceeds the threshold — it does **not** use month-end projections — and if you create the alarm when charges already exceed the threshold, it immediately enters the ALARM state. AWS Budgets, by contrast, can alert on **forecast** amounts.

Beyond monitoring, you can **design for cost reduction**. With cost tools in hand, a good strategy combines several optimization principles, drawing on services such as Amazon RDS, Amazon DynamoDB, AWS Lambda, AWS Budgets, and AWS Trusted Advisor.

TABLE 10.2 Cost-reduction design principles

PRINCIPLE	WHAT TO DO
Set budgets	Use AWS Budgets to set custom cost and usage budgets so you adhere to pay-for-what-you-need
Right-size instances	Use T2 or T3 burstable instances for workloads that occasionally burst (T3 offers up to ~30% better price-to-performance than T2); buy Reserved Instances for long-running instances; use Spot Instances for batch jobs, tuning bids with Spot history reports
Go serverless	Consider AWS Lambda — run code without managing servers and pay only for compute time consumed; you are not charged when your code is not running
Use managed services	Managed services such as Amazon S3 and Amazon RDS remove operational burden, run at cloud scale, and lower cost per transaction
Ask Trusted Advisor	Use AWS Trusted Advisor for specific advice on how to reduce cost

Finding and eliminating waste is its own technique: because it is so easy to spin up resources, unneeded ones are often left running. Use **Amazon CloudWatch metrics and alarms** to find long-running idle instances (a utilization

graph reveals instances that are idle or rarely used) and shut them down, and use **AWS Cost Explorer** to find the costs associated with entire projects or initiatives so you can prioritize the most expensive ones for review.

STOPINATOR SCRIPTS

- A **stopinator** is a generic term for any script or application that looks for and stops unused instances — a best practice for reducing cost.
- They typically run in the **evenings and on weekends**, and are a useful dual-function utility because they can also **start** resources when you need them (for example, at the start of the workday).
- You do **not** need an EC2 instance to run one. A simple, efficient **serverless** design implements the stop/start logic as an **AWS Lambda** function, triggered by an **Amazon CloudWatch Events** event on the desired schedule.

COMMON PITFALL

Cost Explorer and the Cost and Usage Report are not interchangeable. **Cost Explorer** is for visualizing and analyzing **trends** (with forecasting and RI recommendations); the **Cost and Usage Report** exists to deliver the **most granular** line-item data. When a question asks for the most detailed cost breakdown, the answer is the Cost and Usage Report.

10.4 AWS Trusted Advisor

AWS Trusted Advisor is an online resource that helps you reduce cost, increase performance, and improve security by optimizing your AWS environment. It scans your environment and compares it against more than **50 AWS best practices** across **five categories**, then provides recommended actions — many with links that let you take direct action. Its guidance is **real-time**, helping you provision resources according to AWS best practices.

TABLE 10.3 Trusted Advisor recommendation categories

CATEGORY	WHAT IT ADVISES ON
Cost optimization	Save money by eliminating unused and idle resources, or by committing to reserved capacity
Performance	Improve service performance — check service limits, take advantage of provisioned throughput, monitor for overutilized instances
Security	Improve application security by closing gaps, enabling AWS security features, and examining permissions
Fault tolerance	Increase availability and redundancy using auto scaling, health checks, Multi-AZ deployment, and backups
Service limits	Flags services whose usage exceeds 80% of the service limit

COST OPTIMIZATION CHECKS AND SUPPORT PLANS

- Sample **cost optimization** checks identify **idle resources** (Amazon EC2 and Amazon RDS instances), **underused load balancers and volumes**, and **unused Elastic IP addresses**.
- Use the cost-optimization checks to achieve a **base level** of cost savings.
- **Core** checks and recommendations are available to **all** customers.
- **Additional** checks and recommendations require a **Business or Enterprise** support plan.

COMMON PITFALL

Trusted Advisor is broader than cost. Remember all five categories — cost optimization, performance, security, fault tolerance, and service limits — but note the **service-limits** check specifically flags usage over **80%** of a limit. For pure cost questions, point to the **Cost Optimization** category.

10.5 Exam strategy, lab, and hands-on activity

The module's **test strategies** section reinforces one skill: before you read the answer choices, fully understand the scenario and question, and hunt for the **keywords** that let you rule out distractors and narrow to the correct answer. The Module 10 scenario describes a non-profit with committees on individual accounts that wants to reduce cost, share expenses fairly across committees, and provide use-and-expense details to donors — signals pointing toward AWS Organizations, consolidated billing, and cost-allocation tagging. The actual question, however, is a precise limit.

MODULE 10 SCENARIO QUESTION

A Support Specialist finds that a **service control policy (SCP)** in AWS Organizations has exceeded its maximum size and begins removing white-space characters. *What is the maximum size of an SCP document?* Isolate the keyword — the **maximum size of an SCP** — and the answer is **5,120 bytes**. The other choices (1,510 bytes, 2,048 characters, 10,240 characters, 10,510 bytes) are distractors; note that the limit is measured in **bytes**, not characters.

LAB 7 – MANAGING RESOURCES WITH TAGGING

- **Apply tags** to existing AWS resources.
- **Query** resources based on tags.
- Use **scripting** to terminate and stop tagged resources.
- Additional challenge: **terminate non-compliant instances** using tags — the tag-or-terminate pattern in practice.

ACTIVITY 10 – OPTIMIZING AN EC2 INSTANCE (MOM & POP CAFÉ)

In the café project, Sophie wants to reduce AWS costs after the database migration to Amazon RDS. The optimization has two parts: **uninstall the decommissioned local MariaDB database** from the café EC2 instance (reducing its storage requirements) and **downsize** the instance from a **T2 small** to a **T2 micro** (right-sizing, since the database process no longer runs on it). More generally, there are **two ways to optimize an EC2 instance's utilization**: (1) choose the correct **size for the attached storage volumes** — estimate current needs plus a little room for growth — and (2) select the **right instance type** based on the CPU and memory requirements of the workload. The **AWS Simple Monthly Calculator** can estimate the resulting cost savings.

Module summary

- A tag is a key-value label you attach to an AWS resource to identify and categorize it; you assign it after creation, keys/values are case-sensitive, you get up to 50 per resource, and reserved aws:-prefix tags can't be edited and don't count toward the limit.
- Good tags follow organizationally relevant dimensions (Environment, Application, Owner, Department/Cost center, Purpose, Stack) that support inventory, access control, cost tracking, automation, and organization.

- Enforce tags two ways: an IAM policy Condition element (aws:RequestTag, aws:TagKeys) prevents non-compliant creation, and the AWS Config required-tags managed rule detects and reports untagged resources afterward.
- Tag-driven automation includes scheduled start/stop of tagged instances, the tag-or-terminate (Conformity Monkey) compliance pattern, cost-allocation billing reports, and the Resource Groups Tagging API for programmatic tag management.
- The core cloud cost lever is pay-only-for-what-you-need: turn off dev/test, after-hours, disaster-recovery, and temporary resources, using stopinator scripts — ideally serverless via AWS Lambda + CloudWatch Events, with no EC2 instance.
- Four cost tools: AWS Budgets (thresholds and actual-or-forecast alerts), Cost Explorer (trends, 13 months history, 3-month forecast, .csv export), the Cost and Usage Report (most granular data), and CloudWatch billing alarms (actual estimated charges, created in us-east-1); the Billing Dashboard shows month-to-date status.
- Design for cost reduction by setting budgets, right-sizing (T2/T3 burstable, Reserved Instances for long-running, Spot for batch), going serverless with Lambda, using managed services, and eliminating waste with CloudWatch metrics and Cost Explorer.
- AWS Trusted Advisor scans your environment against 50+ best practices in five categories — cost optimization, performance, security, fault tolerance, and service limits (flags usage over 80% of a limit); core checks are free, the full set needs Business/Enterprise support.
- Exam craft: read the scenario for keywords first; the module scenario's SCP maximum size is 5,120 bytes. Lab 7 applies, queries, and scripts against tags; Activity 10 optimizes an EC2 instance by right-sizing storage and instance type.

Review questions

1. **What is a tag, and name three characteristics that constrain how you use one.**

   Answer: A tag is a key-value label you attach to an AWS resource to identify and categorize it. Characteristics: it can be assigned only after the resource exists; keys and values are case-sensitive; you can have up to 50 per resource; and aws:-prefix tags are AWS-reserved (you can't edit/delete them and they don't count toward the 50).

2. **How do IAM and AWS Config each help enforce tagging, and how do their roles differ?**

   Answer: An IAM policy uses the Condition element (aws:RequestTag, aws:TagKeys) to prevent creation of a resource that lacks the required tags — it is preventive. AWS Config's required-tags managed rule detects and reports resources already missing a required tag key/value — it is detective. A good strategy uses both.

3. **List some cost benefits/strategies the cloud provides for turning off unused resources.**

   Answer: You pay only for what you need, when you need it, so you can shut down development/test environments at project end, specific services after business hours and during holidays, disaster-recovery environments until needed, and temporary tagged instances after their period — often via scripts or templates that stop/start whole environments.

4. **Compare AWS Budgets, Cost Explorer, the Cost and Usage Report, and CloudWatch billing alarms.**

   Answer: AWS Budgets sets thresholds and alerts on actual or forecast spend by period. Cost Explorer visualizes and analyzes trends (13 months history, 3-month forecast, RI recommendations, .csv export). The Cost and Usage Report delivers the most granular line-item data. A CloudWatch billing alarm alerts when actual estimated charges exceed a threshold and must be created in us-east-1.

5. Why must a CloudWatch billing alarm be created in us-east-1, and does it use forecasts?

Answer: Billing metric data is stored centrally in us-east-1 and represents worldwide charges, so the alarm must be created there. It triggers only when actual estimated charges exceed the threshold — it does not use month-end projections (unlike AWS Budgets, which can alert on forecasts).

6. What is a stopinator, and how is a serverless stopinator built?

Answer: A stopinator is a generic script/application that finds and stops (and can start) unused instances on a schedule to cut cost, typically running evenings and weekends. A serverless stopinator needs no EC2 instance: the stop/start logic is an AWS Lambda function triggered by an Amazon CloudWatch Events event on a schedule.

7. What is AWS Trusted Advisor, and what are its five recommendation categories?

Answer: Trusted Advisor is an online resource that helps reduce cost, increase performance, and improve security by scanning your environment against 50+ AWS best practices and recommending actions with real-time guidance. Its five categories are cost optimization, performance, security, fault tolerance, and service limits (which flags usage over 80% of a limit).

8. In Activity 10, what two ways optimize an EC2 instance's utilization?

Answer: Choose the correct size for the attached storage volumes (estimate current needs plus a little growth), and select the right instance type based on the workload's CPU and memory requirements. In the café case this meant uninstalling the decommissioned local MariaDB database and downsizing from a T2 small to a T2 micro.