Tagging Best Practices

AWS Whitepaper

Tagging Best Practices: AWS Whitepaper

Table of Contents

Tagging Best Practices

Publication date: **February 24, 2023** (*Document revisions* (p. 33))

Amazon Web Services (AWS) allows you to assign metadata to many of your AWS resources in the form of tags. Each tag is a simple label consisting of a key and an optional value to store information about the resource or data retained on that resource. This whitepaper focuses on tagging use cases, strategies, techniques, and tools that can help you to categorize resources by purpose, team, environment, or other criteria relevant to your business. Implementing a consistent tagging strategy can make it easier to filter and search for resources, monitor cost and usage, as well as manage your AWS environment.

This paper builds on the practices and guidance provided in the Organizing Your AWS Environment Using Multiple Accounts whitepaper. It is recommended that you read that whitepaper before this one. AWS recommends that you establish your cloud foundation in a holistic way. For additional information, refer to Establishing your Cloud Foundation on AWS.

Are you Well-Architected?

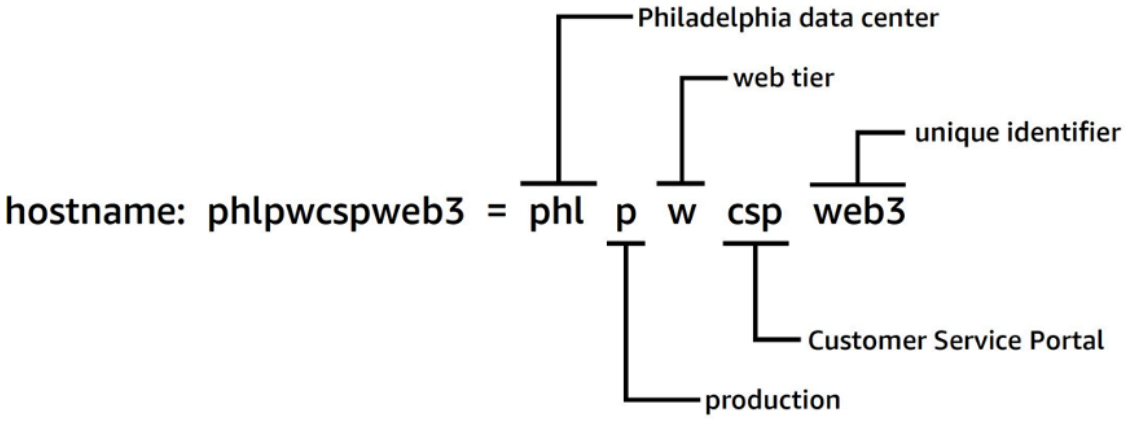
The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

AWS makes it easy to deploy your workloads in AWS by creating resources, such as Amazon EC2 instances, Amazon EBS volumes, security groups, and AWS Lambda functions. You can also scale and grow the fleet of AWS resources that hosts your applications, stores your data, and expands your AWS infrastructure over time. As your AWS usage grows to many resource types spanning multiple applications, you will need a mechanism to track which resources are assigned to which application. Use this mechanism to support your operational activities, such as cost monitoring, incident management, patching, backup, and access control.

In on-premises environments, this knowledge is often captured in knowledge management systems, document management systems, and on internal wiki pages. With a configuration management database (CMDB), you can store and manage the relevant detailed metadata using standard change control processes. This approach provides governance, but requires additional effort to develop and maintain. You can take a structured approach to the naming of resources, but a resource name can only hold a limited amount of information.



Structured approach to resource naming

For example, EC2 instances have a predefined tag called Name that provides similar functionality and allows you to name workloads as they are moved to AWS.

In 2010, AWS launched resource tags to provide a flexible and scalable mechanism for attaching metadata to your resources. This whitepaper guides you through the process of developing and implementing a robust tagging strategy across your AWS environment. This guidance will help you ensure tagging consistency and coverage that supports your decision-making and operational activities

What are tags?

A tag is a *key-value pair* applied to a resource to hold metadata about that resource. Each tag is a label consisting of a key and an optional value. Not all services and resource types currently support tags (see Services that support the Resource Groups Tagging API). Other services may support tags via their own APIs. It should be noted that tags are not encrypted and should not be used to store sensitive data, such as personally identifiable information (PII).

Tags that a user creates and applies to AWS resources using the AWS CLI, API, or the AWS Management Console are known as *user-defined* tags. Several AWS services, such as AWS CloudFormation, Elastic Beanstalk, and Auto Scaling, automatically assign tags to resources that they create and manage. These keys are known as *AWS generated* tags and are typically prefixed with `aws`. This prefix cannot be used in user-defined tag keys.

There are usage requirements and limits on the number of user-defined tags that can be added to an AWS resource. For more information, refer to Tag naming limits and requirements in the *AWS General Reference guide*. AWS generated tags do not count against these user-defined tag limits.

Table 1 – Examples of user-defined tag keys and values

Instance ID	Tag Key	Tag Value
i-01234567abcdef89a	CostCenter	98765
	Stack	Test
i-12345678abcdef90b	CostCenter	98765
	Stack	Production

Table 2 – Examples of AWS generated tags

AWS Generated Tag Keys	Rationale
aws:ec2spot:fleet-request-id	Identifies the Amazon EC2 Spot Instance request that launched the instance
aws:cloudformation:stack-name	Identifies the AWS CloudFormation stack that created the resource
lambda-console:blueprint	Identifies blueprint used as a template for an AWS Lambda function
elasticbeanstalk:environment-name	Identifies the application that created the resource
aws:servicecatalog: provisionedProductArn	The provisioned product Amazon Resource Name (ARN)
aws:servicecatalog:productArn	The ARN of the product from which the provisioned product was launched

AWS generated tags form a namespace. For example, in a CloudFormation template, you define a set of resources to be deployed together in a `stack`, where `stack-name` is a descriptive name that you assign to identify it. If you examine a key such as `aws:cloudformation:stack-name`, you can see the namespace that is used to scope the parameter uses three elements: **aws** the *organization,* **cloudformation** the *service, and* **stack-name** the *parameter.*

User-defined tags can also use namespaces and using an organizational identifier as a prefix is recommended. This helps you quickly identify whether a tag is something from your managed schema, or something defined by a service or tool that you are using in your environment.

In the Establishing Your Cloud Foundation on AWS whitepaper, we recommend a set of tags that should be implemented. It's highly likely that different businesses will have different allowed patterns and different lists for a given tag. Looking at the example in Table 3:

Table 3 – Same tag key, different value validation rules

Organization	Tag Key	Tag Values Validation	Tag Value Example
Company A	`CostCenter`	`5432, 5422, 5499`	`5432`
Company B	`CostCenter`	`ABC*`	`ABC123`

If these two schemas are in separate organizations, then there is no issue with tag conflicts. However, should these two environments merge, then the namespaces can conflict and validation becomes more complex. This scenario might seem unlikely, but businesses are acquired or merged, and there are other scenarios, such as clients working with a managed service provider, game publisher, or venture capital business, where accounts from different organizations are part of a shared AWS Organization. By using the business name as a prefix to define a unique namespace, these challenges can be avoided, as shown in Table 4:

Table 4 – Use of namespaces in tag keys

Organization	Tag Key	Tag Values Validation	Tag Value Example
Company A	`company-a:CostCenter`	`5432, 5422, 5499`	`5432`
Company B	`company-b:CostCenter`	`ABC*`	`ABC123`

In large and complex organizations where businesses are acquired and divested regularly, this situation will occur more frequently. As the new acquisition's processes and practices are harmonized across the wider group, the situation is resolved. Having distinct namespaces helps, as the use of the older tags can be reported on and the relevant teams contacted to adopt the new schema. A namespace can also be used to indicate a scope, represent a use-case, or an area of responsibility that is aligned to organizational owners.

Table 5 – Example of scope or use case scope within tag keys

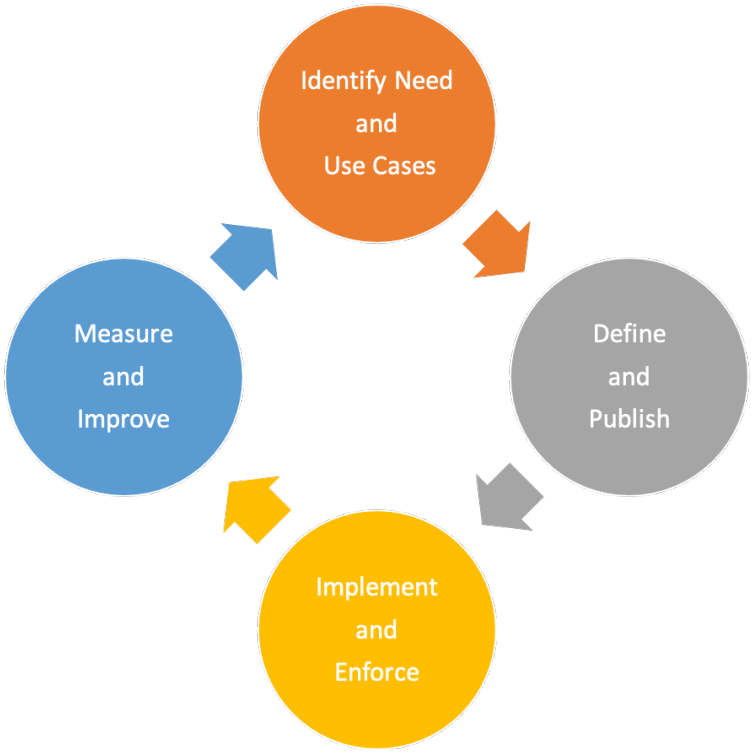
Use Case	Tag Key	Rationale	Allowed Values
Data Classification	`example-inc:info-sec:data-classification`	Information security defined set of data classification	`sensitive`, `company-confidential`, `customer-identifiable`

Use Case	Tag Key	Rationale	Allowed Values
Operations	`example-inc:dev-ops:environment`	Implement scheduling of test and development environments	`development`, `staging`, `quality-assurance`, `production`
Disaster Recovery	`example-inc:disaster-recovery:rpo`	Define the recovery point objective (RPO) for a resource	6h, 24h
Cost Allocation	`example-inc:cost-allocation:business-unit`	Finance teams need cost reporting on each team's usage and spend	`corporate`, `recruitment`, `support`, `engineering`

Tags are simple and flexible. Both the key and the value of the tag are variable-length strings and can support a wide character set. For more information on lengths and character sets, see Tagging AWS resources in the *AWS General Reference*. Tags are case sensitive, meaning that `costCenter` and `costcenter` are different tag keys. In different countries, the spelling of a word might differ, which might affect your keys. For example, in the United States one might define a key as `costcenter`, but in the United Kingdom, `costcentre` might be preferred. These are different keys from the resource-tagging perspective. Define spelling, case, and punctuation as part of your tagging strategy. Use these definitions as a reference for anyone creating or managing resources. This topic is discussed in more detail in the next section, Building your tagging strategy (p. 6).

Building your tagging strategy

As with many practices in operations, implementing a tagging strategy is *a process of iteration and improvement*. Start small with your immediate priority and grow the tagging schema as you need to.



Tagging strategy iteration and improvement cycle

Throughout this process, ownership is key to accountability and progress. As tags can be used for a variety of purposes, the overall tagging strategy can be split into areas of responsibility within an organization. Tagging allows a programmatic approach to activities that depend upon the characterization of resources. The range of stakeholders that can benefit from tagging will depend on the size of the organization and operational practices. Larger organizations can benefit from clearly defining the responsibilities of the teams involved in building and implementing a tagging strategy. Some stakeholders can be responsible for identifying the needs (defining use cases) for tagging, while others can be responsible for maintaining, implementing, and improving the tagging strategy.

By assigning ownership, you are in a good position to implement individual aspects of the strategy. Where appropriate, this ownership can be formalized as policy and documented in a responsibility matrix (for example, RACI: Responsible, Accountable, Consulted, and Informed), or in a shared responsibility model. In smaller organizations, teams might play multiple roles in a tagging strategy, from requirements definition to implementation and enforcement.

Defining needs and use cases

Start building your strategy by engaging with stakeholders who have a fundamental underlying need to consume metadata. These teams define the metadata that resources need to be tagged with to support their activities, such as reporting, automation, and data classification. They outline how the resources need to be organized and which policies they need to be mapped to. Examples of roles and functions that these stakeholders can have in organizations include:

- **Finance** and **Line of Business** need to understand the value of investment by mapping it to costs to prioritize actions that need to be taken when addressing inefficiency. Understanding *cost vs value generated* helps to identify unsuccessful lines of business or product offerings. This leads to informed decisions about continuing support, adopting an alternative (for example, using a SaaS offering or managed service), or retiring an unprofitable business offering.
- **Governance** and **Compliance** need to understand the categorization of data (for example, public, sensitive, or confidential), whether a specific workload is in or out of scope for audit against a specific standard or regulation and the criticality of the service (whether the service or application is business-critical) to apply appropriate controls and oversight, such as permissions, policies, and monitoring.
- **Operations** and **Development** need to understand the workload lifecycle and implemented stages of their supported products. Management of release stages, (for example, Development, Test, Production split) and their associated support prioritizations and stakeholder management requirements. Duties such as Backups, Patching, Observability and Deprecation also need to be defined and understood.
- **Information Security (InfoSec)** and **Security Operations (SecOps)** outline what controls must be applied and which are recommended. InfoSec normally defines the implementation of the controls, and SecOps is generally responsible for managing those controls.

Depending on your use case, priorities, size of your organization, and operational practices, you might need representation from various teams within the organization, such as Finance (including Procurement), Information Security, Cloud Enablement, and Cloud Operations. You also need representation from application and process owners for functions such as patching, backup and restore, monitoring, job scheduling, and disaster recovery. These representatives help drive the definition, implementation, and measure the effectiveness of the tagging strategy. They should *work backwards* from stakeholders and their use cases, and conduct a cross-functional workshop. In the workshop, they get a chance to share their perspectives and needs, and help drive an overall strategy. Examples of participants and their involvement in various use cases are described later in this whitepaper.

The stakeholders also define and validate the keys for mandatory tags, and can recommend the scope for optional tags. For example, Finance Teams might need to relate a resource to an internal cost center, business unit, or both. Thus, they might require that certain tag keys, such as `CostCenter` and `BusinessUnit`, be made mandatory. Individual development teams might decide to use additional tags for automation purposes, such as `EnvironmentName`, `OptIn`, or `OptOut`.

Key stakeholders need to agree on the tagging strategy approach, and document the answers for compliance- and governance-related questions, such as:

- What use cases need to be addressed?
- Who is responsible for tagging resources (implementation)?
- How are tags enforced and what methods and automation will be used (proactive vs reactive)?
- How are tagging effectiveness and goals measured?
- How often should the tagging strategy be reviewed?
- Who drives improvements? How is this done?

Business functions, such as Cloud Enablement, Cloud Business Office, and Cloud Platform Engineering, can then play a role of facilitators for the process of building the tagging strategy, help drive its adoption, and ensure consistency of its application by measuring progress, removing roadblocks, and reducing duplicated effort.

Defining and publishing a tagging schema

Employ a consistent approach in tagging your AWS resources, both for mandatory and optional tags. A comprehensive tagging schema helps you to achieve this consistency. The following examples can help get you started:

- Agree on the mandatory tag keys
- Define acceptable values and tag naming conventions (upper or lower case, dashes or underscores, hierarchy, and so on)
- Confirm values would not constitute personally identifiable information (PII)
- Decide who can define and create new tag keys
- Agree on how to add new mandatory tag values and how to manage optional tags

Review the following tagging categories table, which can be used as a baseline, of what you might include in you tagging schema, you would still need to determine the convention you will use for the tag key and what values are permitted for each. The tagging schema is the document in which you define this for your environment.

Table 6 – Example of a definitive tagging schema (part 1)

Use Case	Tag Key	Rationale	Allowed Values (Listed or value prefix/suffix)	Used for Cost Allocation	Resource Types	Scope	Required
Cost Allocation	`example-inc:cost-allocation:ApplicationId`	Track *cost vs value generated* by each line of business	`DataLakeX`, `RetailSiteX`	Y	All	All	Mandatory
Cost Allocation	`example-inc:cost-allocation:BusinessUnitId`	Monitor costs by business unit	`Architecture`, `DevOps`, `Finance`	Y	All	All	Mandatory
Cost Allocation	`example-inc:cost-allocation:CostCenter`	Monitor costs by cost center	`123-*`	Y	All	All	Mandatory
Cost Allocation	`example-inc:cost-allocation:Owner`	Which budget holder is responsible for this workload	`Marketing`, `RetailSupport`	Y	All	All	Mandatory
Access Control	`example-inc:access-control:LayerId`	Identify SubComponent / Layer to grant access to resources based on the role	`DB_Layer`, `Web_Layer`, `App_Layer`	N	All	All	Optional
Automation	`example-inc:automation:EnvironmentId`	Implement scheduling of test and development environments, also refered to as software development lifecycle (SDLC) stage	`Prod`, `Dev`, `Test`, `Sandbox`	N	EC2, RDS, EBS	All	Mandatory

Table 6 – Example of a definitive tagging schema (part 2)

Use Case	Tag Key	Rationale	Allowed Values (Listed or value prefix/suffix)	Used for Cost Allocation	Resource Types	Scope	Required
DevOps	`example-inc:operations:Owner`	Which team/squad is responsible for the creation and maintenance of the resource	`Squad01`	N	All	All	Mandatory
Disaster Recovery	`example-inc:disaster-recovery:rpo`	Define the recovery point objective(RPO) for a resource	6h, 24h	N	S3, EBS	Prod	Mandatory
Data Classification	`example-inc:data:classification`	Classify data for compliance and governance	`Public`, `Private`, `Confidential`, `Restricted`	N	S3, EBS	All	Mandatory
Compliance	`example-inc:compliance:framework`	Identifies the compliance framework the workload is subject to	`PCI-DSS`, `HIPAA`	N	All	Prod	Mandatory

After the tagging schema is defined, manage the schema in a version-controlled repository that is made accessible to all the relevant stakeholders for easy reference and trackable updates. This approach improves efficiency and allows for agility.

AWS Organizations – Tag policies

Policies in AWS Organizations allow you to apply additional types of governance to AWS accounts in your organization. A *tag policy* is how you can express your tagging schema in JSON form so that the platform can report and optionally enforce the schema within your AWS environment. The tag policy defines the values that are acceptable for a tag key on specific resource types. This policy can be in the form of a list of values, or a prefix followed by a wildcard character (*). The simple prefix approach is less rigorous than a discrete list of values but requires less maintenance.

The following examples show how to define a tagging policy to validate the values that are acceptable for a given key. Working from the human-friendly tabular definition of the schema, you can transcribe this information into one or more tag policies. Separate policies can be used to support delegated ownership or some policies might only apply in specific scenarios.

ExampleInc-CostAllocation.json

The following is an example of a tag policy that reports on Cost Allocation tags:

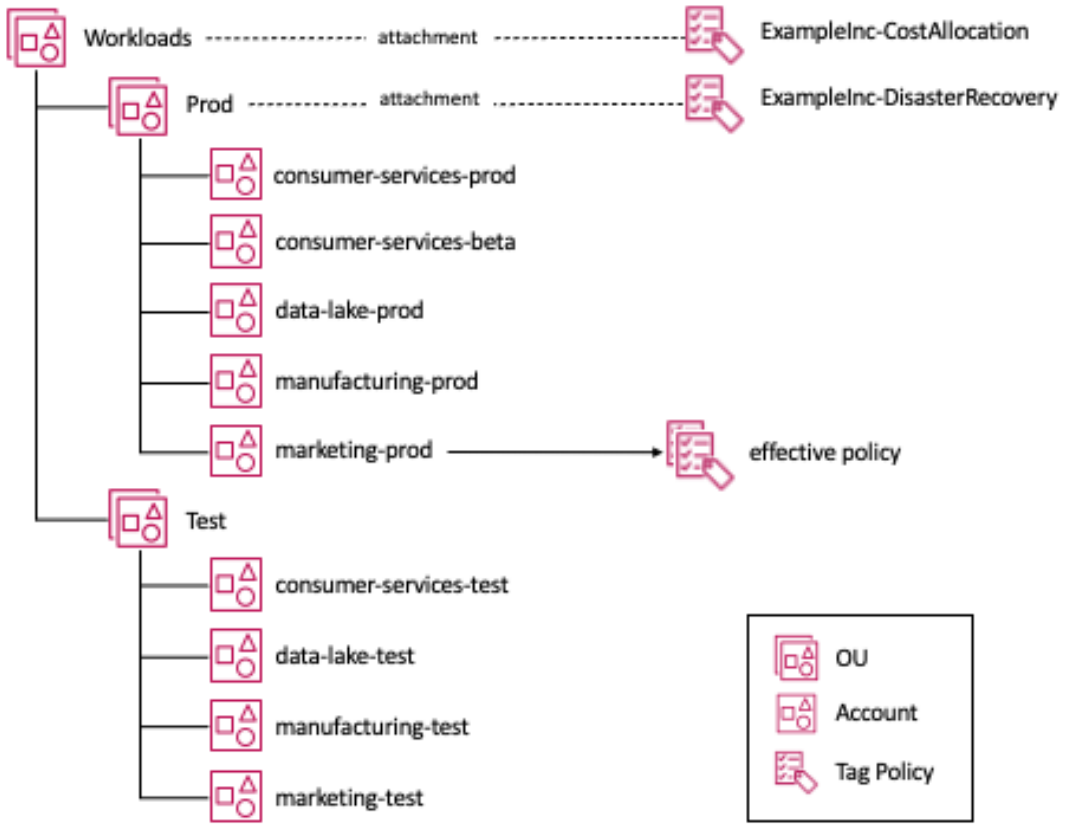
```
{
  "tags": {
    "example-inc:cost-allocation:ApplicationId": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:ApplicationId"
      },
      "tag_value": {
        "@@assign": [
          "DataLakeX",
          "RetailSiteX"
        ]
      }
    },
    "example-inc:cost-allocation:BusinessUnitId": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:BusinessUnitId"
      },
      "tag_value": {
        "@@assign": [
          "Architecture",
          "DevOps",
          "FinanceDataLakeX"
        ]
      }
    },
    "example-inc:cost-allocation:CostCenter": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:CostCenter"
      },
      "tag_value": {
        "@@assign": [
          "123-*"
        ]
      }
    }
  }
}
```

ExampleInc-DisasterRecovery.json

The following is an example of a tag policy that reports on Disaster Recovery tags:

```
{
    "tags": {
        "example-inc:disaster-recovery:rpo": {
            "tag_key": {
                "@@assign": "example-inc:disaster-recovery:rpo"
            },
            "tag_value": {
                "@@assign": [
                    "6h",
                    "24h"
                ]
            }
        }
    }
}
```

In this example, the `ExampleInc-CostAllocation` tag policy is attached to the `Workloads` OU, and therefore applies to all the accounts in both the `Prod` and `Test` child OUs. Similarly, the `ExampleInc-DisasterRecovery` tag policy is attached to the `Prod` OU and therefore only applies to accounts below this OU. The Organizing Your Environment Using Multiple Accounts whitepaper explores the recommended OU structures in more detail.



Attachment of tag policies to an OU structure

Looking at the `marketing-prod` account in the diagram, both tag policies apply to this account, so we have the concept of an *effective policy*, which is the convolution of the policies of a given type that apply to an account. If you primarily manage your resources manually, then you can review the effective policy by visiting the Resource Groups & Tag Editor:Tag policies in the console. If you use infrastructure as code (IaC) or scripting to manage your resources, you can use the `AWS::Organizations::DescribeEffectivePolicy` API call.

Implementing and enforcing tagging

In this section, we'll introduce you to the tools available for the following resource management strategies: manual, infrastructure as code (IaC), and continuous integration/continuous delivery (CI/CD). The key dimension for these approaches is an increasingly frequent rate of deployment.

Manually managed resources

These are typically workloads that fall into the foundation or migration stages of adoption. Often, these are simple largely static workloads that have been built using traditional written procedures or those migrated *as is* using tools such as CloudEndure from an on-premises environment. Migration tools, such as Migration Hub and CloudEndure, can apply tags as part of the migration process. However, if tags were not applied during the original migration or the tagging schema has changed since then, the Tag Editor (a feature of the AWS Management Console) allows you to search for resources using a variety of search criteria and add, modify, or delete tags in bulk. Search criteria can include resources with or without the presence of a particular tag or value. The AWS Resource Tagging API allows you to perform these functions programmatically.

As these workloads are modernized, resource types such as Auto Scaling groups are introduced. These resource types allow for greater elasticity and improved resilience. The auto scaling group manages EC2 instances on your behalf, however, you might still want the EC2 instances to be tagged consistently with the manually created resources. An EC2 launch template provides the means to specify the tags that the Auto Scaling should apply to instances that it creates.

When the resources of a workload are being managed manually, it can be helpful to automate the tagging of resources. There are various solutions available. Automatically tag new AWS resources based on identity or role describes one approach that detects resource creation events using CloudWatch and triggers an AWS Lambda function to retrieve the tags from an AWS Systems Manager Parameter Store, and then applies the tags to the resource. Another approach is to use AWS Config Rules, which can check for `required_tags` and then start a Lambda function to apply them. AWS Config Rules is described in more detail later in this whitepaper.

Infrastructure as code (IaC) managed resources

AWS CloudFormation provides a common language for provisioning all the infrastructure resources in your AWS environment. CloudFormation templates are JSON or YAML files that create AWS resources in an automated manner. When you create AWS resources using CloudFormation templates, you can use the CloudFormation Resource Tags property to apply tags to supported resource types upon creation. Managing the tags as well as the resources with IaC helps ensure consistency.

When resources are created by AWS CloudFormation, the service automatically applies a set of AWS defined tags to the resources created by the CloudFormation template. These are:

```
aws:cloudformation:stack-name
aws:cloudformation:stack-id
aws:cloudformation:logical-id
```

You can easily define a resource group based on the CloudFormation stack. These AWS defined tags are inherited by the resources created by the stack. However, for EC2 instances within an Auto Scaling group, the AWS::AutoScaling::AutoScalingGroup TagProperty needs to be set in the definition of the Auto Scaling group in your CloudFormation template. Alternatively, if you are using an EC2 Launch Template with the Auto Scaling group then you can define the tags in its definition. Using EC2 Launch Templates with Auto Scaling groups or with an AWS container service is recommended. These services can help ensure consistent tagging of your EC2 instances and also support Auto Scaling Across Multiple Instance Types & Purchase Options, which can improve resilience and optimize your compute costs.

AWS CloudFormation Hooks provide developers with a means of keeping key aspects of their application consistent with their organization's standards. Hooks can be configured to provide a *warning* or *prevent deployment*. This feature is best suited to checking key configuration elements in your templates, such as whether an Auto Scaling group is configured to apply customer defined tags to all the EC2 instances it will launches, or to ensure that all S3 buckets are created with the required encryption settings. In both cases, evaluation of this compliance is being pushed to the earlier in the deployment process, with AWS CloudFormation hooks, prior to deployment.

AWS CloudFormation provides the capability to detect when a resource (see Resources that support drift detection) provisioned from a template has been modified and resources no longer match their expected template configurations, this is known as *drift*. If you use automation to apply tags to resources managed via IaC, then you are modifying them, introducing drift. When using IaC, it's currently recommended to manage any tagging requirements as part of the IaC templates, implement AWS CloudFormation hooks and publishing AWS CloudFormation Guard rule sets that can be used by automation.

CI/CD pipeline managed resources

As the maturity of a workload increases, it's likely that techniques such as continuous integration and continuous deployment are adopted. These techniques help to reduce deployment risk by making it easier to deploy small changes more frequently with increased automation of testing. An observability strategy that detects unexpected behavior introduced by a deployment can automatically roll back the deployment with minimal user impact. As you get to the stage of deploying tens of times a day, applying tags retrospectively is simply no longer practical. Everything needs to be expressed as code or configuration, version controlled, and, wherever possible, tested and evaluated before deployment into production. In the combined develop and operations (DevOps) model, many of the practices address operational considerations as code and validate them early in the deployment lifecycle.

Ideally, you want to push these checks as early in the process as you can (as shown with AWS CloudFormation hooks), so that you can be confident that your CloudFormation template meets your policies, before they leave the developer's machine.

AWS CloudFormation Guard 2.0 provides the means to write preventative compliance rules, for anything you can define with CloudFormation. The template is validated against the rules in the development environment. Clearly, this feature has a range of applications but in this whitepaper we will just look at some examples that would ensure the AWS::AutoScaling::AutoScalingGroup TagProperty is always being used.

The following is an example of a CloudFormation Guard rule:

```
let all_asgs = Resources.*[ Type == 'AWS::AutoScaling::AutoScalingGroup' ]

rule tags_asg_automation_EnvironmentId when %all_asgs !empty {
    let required_tags = %all_asgs.Properties.Tags.*[
        Key == 'example-inc:automation:EnvironmentId' ]
    %required_tags[*] {
        PropagateAtLaunch == 'true'
        Value IN ['Prod', 'Dev', 'Test', 'Sandbox']
        <<Tag must have a permitted value
          Tag must have PropagateAtLaunch set to 'true'>>
    }
```

```
}

rule tags_asg_costAllocation_CostCenter when %all_asgs !empty {
    let required_tags = %all_asgs.Properties.Tags.*[
        Key == 'example-inc:cost-allocation:CostCenter' ]
    %required_tags[*] {
        PropagateAtLaunch == 'true'
        Value == /^123-/
        <<Tag must have a permitted value
          Tag must have PropagateAtLaunch set to 'true'>>
    }
}
```

In the code example, we filter the template for all resources that are of the type `AutoScalingGroup` and then have two rules:

- **`tags_asg_automation_EnvironmentId`:** Checks that a tag with this key exists, has a value within the allowed list of values, and that `PropagateAtLaunch` is set to `true`.
- **`tags_asg_costAllocation_CostCenter`:** Checks that a tag exists with this key, has a value that begins with the defined prefix value, and that `PropagateAtLaunch` is set to `true`.

Enforcement

As described previously, **Resource Groups & Tag Editor** provides the means to identify where your resources fail to meet the tagging requirements defined in the tag policies applied to the OUs of the organization. Accessing the **Resource Groups & Tag Editor** console tool from within an Organization member account shows you the policies that apply to that account and the resource within the account that fail to meet the tag policy's requirements. If accessed from the management account (and if **Tag policies** has *Access enabled* in services under AWS Organizations), then it is possible to view the tag policy compliance for all the linked accounts in the organization.

Within the Tag Policy itself, you can enable enforcement for specific resource types. In the following policy example, we've added enforcement such that all resources of types `ec2:instance` and `ec2:volume` are required to be compliant with the policy. There are some known limitations, such as there must be a tag on a resource for it to be evaluated by the tag policy. See Resources that support enforcement with tag policies for a list.

ExampleInc-Cost-Allocation.json

The following is an example of a tag policy that reports and/or enforces Cost Allocation tags:

```
{
  "tags": {
    "example-inc:cost-allocation:ApplicationId": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:ApplicationId"
      },
      "tag_value": {
        "@@assign": [
          "DataLakeX",
          "RetailSiteX"
        ]
      },
      "enforced_for": {
        "@@assign": [
          "ec2:instance",
          "ec2:volume"
        ]
      }
```

```
    },
    "example-inc:cost-allocation:BusinessUnitId": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:BusinessUnitId"
      },
      "tag_value": {
        "@@assign": [
          "Architecture",
          "DevOps",
          "FinanceDataLakeX"
        ]
      },
      "enforced_for": {
        "@@assign": [
          "ec2:instance",
          "ec2:volume"
        ]
      }
    },
    "example-inc:cost-allocation:CostCenter": {
      "tag_key": {
        "@@assign": "example-inc:cost-allocation:CostCenter"
      },
      "tag_value": {
        "@@assign": [
          "123-*"
        ]
      },
      "enforced_for": {
        "@@assign": [
          "ec2:instance",
          "ec2:volume"
        ]
      }
    }
  }
}
```

AWS Config (`required_tag`)

AWS Config is a service that allows you to assess, audit, and evaluate the configurations of your AWS resources (see Resource types supported by AWS Config). In the case of tagging, we can use it to identify resources that are lacking tags with specific keys, using the `required_tags` rule (refer to Resource types supported by required_tags). From the earlier example, we might test for the existence of the key on all EC2 instances. In cases where the key doesn't exist, the instance will be registered as non-compliant. This AWS CloudFormation template describes an AWS Config Rule to test for the presence of the mandatory keys describe in the table, on S3 buckets, EC2 instances, and EBS volumes.

```
Resources:
 MandatoryTags:
   Type: AWS::Config::ConfigRule
   Properties:
     ConfigRuleName: ExampleIncMandatoryTags
     Description: These tags should be in place
     InputParameters:
       tag1Key: example-inc:cost-allocation:ApplicationId
       tag2Key: example-inc:cost-allocation:BusinessUnitId
       tag3Key: example-inc:cost-allocation:CostCenter
       tag4Key: example-inc:automation:EnvironmentId
     Scope:
       ComplianceResourceTypes:
       - "AWS::S3::Bucket"
       - "AWS::EC2::Instance"
       - "AWS::EC2::Volume"
```

```
      Source:
        Owner: AWS
        SourceIdentifier: REQUIRED_TAGS
```

For environments where resources are managed manually, an AWS Config rule can be enhanced to automatically add the missing tag key to the resources using an automated remediation via an AWS Lambda function. While this works well for static workloads, it's progressively less effective as you start managing your resources via IaC and deployment pipelines.

AWS Organizations – Service control policies (SCPs) are a type of organization policy that you can use to manage permissions in your organization. SCPs offer central control over the maximum available permissions for all accounts in your organization or organizational unit (OU). SCPs only affect users and roles that are managed by accounts that are part of the organization. Although they do not affect resources directly, they restrict the permissions of users and roles which includes the permissions for tagging actions. With regards to tagging, SCPs can provide additional granularity for tag enforcement in addition to what tag policies can provide.

In the following example, the policy will deny `ec2:RunInstances` requests where the `example-inc:cost-allocation:CostCenter` tag is not present.

The following is a deny SCP:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyRunInstanceWithNoCostCenterTag",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": [
        "arn:aws:ec2:*:*:instance/"
      ],
      "Condition": {
        "Null": {
          "aws:RequestTag/example-inc:cost-allocation:CostCenter": "true"
        }
      }
    }
  ]
}
```

It is not possible to retrieve the effective service control policy that applies to a linked account by design. Where you enforce tagging with SCPs, documentation needs to be available to developers so they can ensure their resources meet the policies that have been applied to their accounts. Providing read only access to CloudTrail events within their account can support developers in the task of debugging when their resources fail to comply.

Measuring tagging effectiveness and driving improvements

After you have implemented a tagging strategy, it's important to measure its effectiveness against the target use cases. The measure of effectiveness will vary by use case. For example:

- *Cost attribution.* You could measure the tagging coverage of resources based on spend using tools such as AWS Cost Explorer or AWS Cost and Usage Report. For example, you could track the *% of tagged vs untagged* resources that generate charges, particularly monitoring specific tag keys.

- *Automation.* You might want to audit if the desired result has been achieved. For example, whether non-production EC2 instances are suspended outside of business hours, auditing instance start and stop times.

Resource Groups & Tag Editor within the management account, provides additional capabilities to analyze tag policy compliance for all the linked accounts in your organization.

Based on the results of the measurement of your tagging effectiveness, identify if any improvements or alterations are needed in any of the steps such as use case definition, tagging schema implementation or enforcement. Make necessary changes and repeat the cycle until desired effectiveness is achieved. In the example with cost attribution, you can look at percentage improvement.

Since it's the developers and operators that need to perform the actual tagging of resources, it's critical to have them take ownership. This isn't the only additional responsibility that teams typically assume in their journey of AWS adoption. Increased responsibility for the security and cost of developing and operating their application are also important. Organizations often use goals and targets as means of motivating the adoption of new practices, so this can also apply here.

Tagging use cases

Topics

Tags for cost allocation and financial management

One of the first tagging use cases organizations often tackle is visibility and management of cost and usage. There are usually a few reasons for this:

- **It's typically a well understood scenario and requirements are well known.** For example, finance teams want to see the total cost of workloads and infrastructure that span across multiple services, features, accounts, or teams. One way to achieve this cost visibility is through consistent tagging of resources.
- **Tags and their values are clearly defined.** Usually, cost allocation mechanisms already exist in an organization's finance systems, for example, tracking by cost center, business unit, team, or organization function.
- **Rapid, demonstrable return on investment.** It's possible to track cost optimization trends over time when resources are tagged consistently, for example, for resources that were rightsized, auto-scaled, or put on a schedule.

Understanding how you incur costs in AWS allows you to make informed financial decisions. Knowing where you have incurred costs at the resource, workload, team, or organization level enhances your understanding of the value delivered at the applicable level when compared to the business outcomes achieved.

The engineering teams might not have experience with financial management of their resources. Attaching a person with a specialized skill in AWS financial management who can train engineering and development teams on the basics of AWS financial management, and create a relationship between finance and engineering to foster the culture of FinOps will help achieve measurable outcomes for the business, and encourage teams to build with cost in mind. Establishing good financial practices is covered in depth by the Cost Optimization Pillar of the Well-Architected Framework, but we will touch on a few of the fundamental principles in this whitepaper.

Cost allocation tags

Cost allocation refers to the assignment or distribution of incurred costs to the users or beneficiaries of those costs following a defined process. In the context of this whitepaper, we divide cost allocation into two types: *showback* and *chargeback*.

Showback tools and mechanisms help increase cost awareness. *Chargeback* helps with cost recovery and drives enablement of cost awareness. *Showback* is about presentation, calculation, and reporting of charges incurred by a specific entity, such as business unit, application, user, or cost center. For example: "the infrastructure engineering team was responsible for $X of AWS spend last month". *Chargeback* is about an actual charging of incurred costs to those entities via an organization's internal accounting processes, such as financial systems or journal vouchers. For example: "$X was deducted from the infrastructure engineering team's AWS budget". In both cases, tagging resources appropriately can help

allocate the cost to an entity, the only difference being whether or not someone is expected to make a payment.

Your organization's financial governance might require transparent accounting of costs incurred at the application, business unit, cost center, and team level. Performing cost attribution supported by Cost Allocation Tags provides you the data necessary to accurately attribute the costs incurred by an entity from appropriately tagged resources.

- **Accountability** — Ensure that cost is allocated to those who are responsible for resource usage. A single point of service / group can be accountable for spend reviews and reporting.
- **Financial transparency** — Show a clear view into cash allocations towards IT by creating effective dashboards and meaningful cost analysis for leadership.
- **Informed IT investments** — Track ROI based on project, application, or business line, and empower teams to make better business decisions, for example, allocate more funding to revenue generating applications.

In summary, cost allocation tags can help to tell you:

- Who owns the spend and is responsible for optimizing it?
- What workload, application, or product is incurring the spend? Which environment or stage?
- What spend areas are growing fastest?
- How much spend can be deducted from an AWS budget based on past trends?
- What was the impact of cost optimization efforts within particular workloads, applications, or products?

Activating resource tags for cost allocation helps with the definition of measurement practices within the organization that can be used to provide the visibility of AWS usage that increases transparency into accountability for spend. It also focuses on creating an appropriate level of granularity with respect to cost and usage visibility and influencing cloud consumption behaviors through cost allocation reporting and KPI tracking.

Building a cost allocation strategy

Defining and implementing a cost allocation model

Create account and cost structure for the resources being deployed in AWS. Establish the relationship between costs from AWS spend, how those costs were incurred, and who or what incurred those costs. Common cost structures are based on AWS Organizations, AWS accounts, environments, and entities within your organizations, such as a line of business or workload. Cost structures can be based on multiple attributes to permit the examination of costs in different ways or at different levels of granularity such as rolling up the costs of individual workloads to the line of business they serve.

When choosing a cost structure that aligns with the desired outcomes, evaluate the cost allocation mechanisms on the *ease of implementation* versus *desired accuracy*. This might include considerations in regards to accountability, tooling availability, and cultural changes. Three popular cost allocation models that AWS customers usually start from are:

- **Account-based** — This model requires the least amount of effort and provides high accuracy for showbacks and chargebacks, and is suitable for organizations that have a defined account structure (and is consistent with the recommendations of the Organizing Your AWS Environment Using Multiple Accounts whitepaper). This provides clear cost visibility on a per-account basis. For cost visibility and allocation, one can use AWS Cost Explorer, Cost and Usage Reports, as well as AWS Budgets for cost monitoring and tracking, as these tools provide filtering and grouping options by AWS accounts. From cost allocation perspective, this model doesn't have to rely on accurate tagging of individual resources.

- **Business Unit or Team-based** — Cost allocatable to teams, business units, or organizations within an enterprise. This model requires a moderate amount of effort, provides high accuracy for showbacks and chargebacks, and is suitable for organizations that have a defined account structure (typically using AWS Organizations) with separation between various teams, applications, and workload types. This provides clear cost visibility across teams and applications, and as additional benefit reduces the risk of hitting AWS service quotas within a single AWS account. For example, each team may have five accounts (`prod`, `staging`, `test`, `dev`, `sandbox`), and no two teams and applications will share the same account. With such structure AWS Cost Categories will then provide the functionality to group accounts and/or other tags ("meta-tagging") into categories, which can be tracked in the tools mentioned in the previous example. It's important to note that AWS Organizations allows tagging of accounts and organizational units (OUs), however these tags will not be applicable for cost allocation and billing reporting (that is, you cannot group or filter your cost in AWS Cost Explorer by OU). AWS Cost Categories should be used for this purpose.
- **Tag-based** — This model requires more effort compared to the previous two and will provide high accuracy for showbacks and chargebacks depending on the requirements and end goal. While we strongly recommend that you adopt the practices outlined in Organizing Your AWS Environment Using Multiple Accounts whitepaper, realistically customers often find themselves with mixed and complex account structures that take time to migrate away from. Implementing a rigorous and effective tagging strategy is the key in this scenario, followed by activating relevant tags for cost allocation in the Billing and Cost Management console (in AWS Organizations, tags can be activated for cost allocation only from the Management Payer account). After tags are activated for cost allocation, then tools for cost visibility and allocation that were mentioned in the previous methods can be used for showbacks and chargebacks. Note that cost allocation tags are not retrospective, and will only appear in billing reporting and cost tracking tools after they were activated for cost allocation.

To summarize, if you need to track costs by business unit, you can use AWS Cost Categories to group linked accounts within AWS Organization accordingly and view this grouping in billing reports. When you create separate accounts for production and non-production environments, you can also filter the costs related to environments in tools such as AWS Cost Explorer, or track those costs using AWS Budgets. Finally, if your use case requires more granular cost tracking, for example, by individual workloads or applications, you can tag resources within those accounts accordingly, activate those tag keys for cost allocation on the management account, and then filter that cost by tag keys in the billing reporting tools.

Establishing cost reporting and monitoring processes

Start with identifying the types of costs that are important for internal stakeholders (for example, daily spend, cost by account, cost by X, amortized costs). By doing so, you can mitigate budgetary risks associated with unexpected or anomalous spend faster than waiting for the finalized AWS invoice. Tags provide the attribution that enables these reporting scenarios. Insights gained from reporting can inform your actions to mitigate the impact from anomalous and unexpected spend on financial budgets. When there is an unexpected surge in costs, it's important to evaluate if there has been an unexpected surge in the value delivered so that you can determine if and what action is required.

When developing a tagging strategy to support cost allocation, keep in mind the following elements:

- **AWS Organizations.** Cost allocation within multiple accounts can be performed by account, groups of accounts, or group of tags created for resources on those accounts. Tags created for resources residing in individual accounts in AWS Organizations can be used for cost allocation only from the management account.
- **AWS Account.** Cost allocation within one AWS account can be performed by additional dimensions such as services or regions. It's possible to further tag resources within an account and work with the groups of such resource tags.
- **Cost Allocation Tags.** Both user-created tags and AWS generated tags can be activated for cost allocation, if necessary. Enabling tags for cost allocation in the billing console (of the management account in AWS Organizations) helps with showbacks and chargebacks.

- **Cost Categories.** AWS Cost Categories allow grouping accounts and grouping tags ("meta-tagging") within an AWS Organization, which further provides capability to analyze the cost related to these categories through tools such as AWS Cost Explorer, AWS Budgets and AWS Cost and Usage Report.

Performing showback and chargeback for business units, teams, or organizations within the enterprise

Attribute costs using your cost allocation process supported by your cost structure and cost allocation tags. Tags can be used to provide showback to teams that are not directly responsible to pay for costs but are responsible for having incurred those costs. This approach provides awareness of their contribution to spend and how those costs are incurred. Perform chargeback to the teams that are directly responsible for costs to recover the expense of the resources they have consumed, and to provide them with awareness of those costs and how they were incurred.

Measuring and circulating efficiency or value KPIs

Agree on a set of unit cost or KPI metrics to measure the impact of your cloud financial management investments. This exercise creates a common language across technology and business stakeholders, and tells an efficiency-based story, rather than a story focused solely on absolute, aggregate spend. For additional information check this blog that talks how unit metrics can help create alignment between business functions.

Allocating unallocatable spend

Depending on the organization's accounting practices, different charge types might require different treatment. Identify the resources or cost categories that cannot be tagged. Depending on the services used and those planned to be used, agree on the mechanisms on how to treat and measure such unallocatable spend. For example, check the list of resources that are supported by AWS Resource Groups and Tag Editor in the *AWS Resource Groups and Tags User Guide*.

A common example of cost category that cannot be tagged is some fees for commitment-based discounts such as Reserved Instances (RI) and Savings Plans (SP). While subscription fees and unused SP and RI fees cannot be tagged in advance of appearing in billing reporting tools, you can track how RI and SP discounts apply to accounts, resources and their tags in AWS Organizations after the fact. For example, in AWS Cost Explorer it's possible to look at the amortized cost, group that spend by the relevant tag keys and apply filters relevant to your use case. In AWS Cost and Usage Report (CUR), you can filter out lines that correspond to usage covered by RI and SP discounts (read more in the use cases section of the CUR documentation) and select the columns that are only relevant to you. Each tag key activated for cost allocation will be presented in its own separate column at the end of the CUR report, similarly to how it's presented in other legacy billing reports, such as monthly cost allocation report. For additional reference, check the AWS Well-Architected Labs for examples of gaining cost and usage insights from CUR data.

Reporting

In addition to AWS tools available to assist with showbacks and chargebacks, there is a range of other AWS created and third-party solutions that can help monitor the cost of tagged resources, and measure the effectiveness of the tagging strategy. Depending on both the requirements and the end objective of the organization, one could either invest time and resources into building customized solutions or purchase tools provided by one of the AWS Cloud Management Tools Competency Partners. If you decide to create your own *single source of truth* cost allocation tool with controlled parameters relevant for the business, AWS Cost and Usage Report (CUR) provides most detailed cost and usage data and enables creation of customized optimization dashboards, allowing filtering and grouping by accounts, services, cost categories, cost allocation tags, and multiple other dimensions. Among CUR-based solutions developed by AWS that can be used as one of these tools, check Cloud Intelligence Dashboards on the AWS Well-Architected Labs website.

Tags for operations and support

An AWS environment will have multiple accounts, resources, and workloads with differing operational requirements. Tags can be used to provide context and guidance to support operations teams to enhance management of your services. Tags can also be also be used to provide operational governance transparency of the managed resources.

Some of the main factors driving consistent definition of operational tags are:

- **To filter resources during automated infrastructure activities.** For example, when deploying, updating or deleting resources. Another, is the scaling of resources for cost optimization and out of hours usage reductions. See AWS Instance Scheduler solution for a working example.
- **Identifying isolated or deprecating resources.** Resources that have exceeded their defined lifespan or have been flagged for isolation by internal mechanisms should be appropriately tagged so as to assist support personnel in their investigation. Deprecating resources should be tagged before isolation, archival and deletion.
- **Support requirements for a group of resources.** Resources often have different support requirements, for example, these requirements could be negotiated between teams or set as part of an applications criticality. Further guidance on operating models can be found in the Operational Excellence Pillar.
- **Enhance the incident management process.** By tagging resources with tags that offer greater transparency in incident management process, support teams and engineers as well as Major Incident Management (MIM) teams can more effectively manage events.
- **Backups.** Tags can also be used to identify the frequency your resources need to be backed up, and where the backup copies need to go or where to restore the backups. Prescriptive guidance for Backup and recovery approaches on AWS.
- **Patching.** Patching mutable instances running in AWS is crucial in both your overarching patching strategy and for the patching of zero-day vulnerabilities. Deeper guidance on the wider patching strategy can be found in the prescriptive guidance . And patching of zero-day vulnerabilities is discussed in this blog.
- **Operational Observability**. Having an operational KPI strategy translated to resource tags will help operations teams to better track whether targets are being met to enhance business requirements. Developing a KPI strategy, is a separate topic but tends to be focused on a business operating in a steady state or where measure the impact and outcomes of change. The KPI Dashboards (AWS Well-Architected labs) and the Operations KPI Workshop (an AWS Enterprise Support proactive service) both address measure performance in a steady state. The AWS enterprise strategy blog article Measuring the Success of Your Transformation, explores KPI measurement for a transformation program, such as IT modernization or migrating from on premises to AWS.

Automated infrastructure activities

Tags can be used in a wide range of automation activities when managing infrastructure. Use of AWS Systems Manager for example will allow you to manage automations and runbooks on resources specified by the defined key-value pair you create. For managed nodes, you could define a set of tags to track or target nodes by operating system and environment, you could then run an update script for all nodes in a group, or review the status of those nodes. Further to this, Systems Manager Resources themselves can also be tagged to further refine and track your automated activities.

Automating the start and stop lifecycle of environment resources can provide a significant cost reduction to any organization. Instance scheduler on AWS is an example of a solution that can start and stop EC2 and RDS instances when they are not required. For example, developer environments utilizing EC2 or RDS instances that are not required to be running on weekends are not utilizing the cost saving potential that the shutting down of those instances can provide. By analyzing the needs of teams and their environments, and properly tagging these resources to automate their management, you can utilize your budget effectively.

An example schedule tag used by instance scheduler on an EC2 instance:

```
{
    "Tags": [
        {
            "Key": "Schedule",
            "ResourceId": "i-1234567890abcdef8",
            "ResourceType": "instance",
            "Value": "mon-9am-fri-5pm"
        }
    ]
}
```

Workload lifecycle

Review accuracy of supporting operational data. Make sure that there are periodic reviews of the tags associated with your workload lifecycle, and that the appropriate stakeholders are involved in these reviews.

Table 7 – Review operational tags as part of the workload lifecycle

Use Case	Tag Key	Rationale	Example Values
Account Owner	`example-inc:account-owner:owner`	The owner of the account and it's contained resources.	`ops-center`, `dev-ops`, `app-team`
Account Owner Review	`example-inc:account-owner:review`	Review of account ownership details being up to date and correct.	<review date in the correct format defined in your tagging library>
Data Owner	`example-inc:data-owner:owner`	The data owner of the accounts residing data.	`bi-team`, `logistics`, `security`
Data Owner Review	`example-inc:data-owner:review`	Review of data ownership details being up to date and correct.	<review date in the correct format defined in your tagging library>

Assigning tags to suspending accounts before migrating to the suspended OU

Before suspending an account and moving into the suspended OU as detailed in the Organizing Your AWS Environment Using Multiple Accounts whitepaper, tags should be added to the account to aid in your internal tracing and auditing of an account's lifecycle. For example, a relative URL or ticket reference on an organization's ITSM ticketing system, that shows the audit trail for an application being suspended.

Table 8 - Add operational tags when workload lifecycle enters new stage

Use Case	Tag Key	Rationale	Example Values
Account Owner	`example-inc:account-owner:owner`	The owner of the account and it's contained resources.	`ops-center`, `dev-ops`, `app-team`

Use Case	Tag Key	Rationale	Example Values
Data Owner	`example-inc:data-owner:owner`	The data owner of the accounts residing data.	`bi-team`, `logistics`, `security`
Suspended Date	`example-inc:suspension:date`	The date that the account was suspended	<suspended date in the correct format defined in your tagging library>
Approval for suspension	`example-inc:suspension:approval`	The link to the approval of account suspension	`workload/deprecation`

Incident management

Tags can play a vital part in all phases of incident management starting from incident logging, prioritization, investigation, communication, resolution to closure.

Tags can detail where an incident should be logged, the team or teams who should be informed of the incident, and the defined escalation priority. It's important to remember that tags are not encrypted, so consider what information you store in them. Also in organizations, teams, reporting lines responsibilities change so consider storing a link to a secure portal where this information can be more effectively managed. These tags don't need to be exclusive. For instance, the application id, could be used to lookup the escalation paths in an IT service management portal. Make sure it is clear in your operational definitions that this tag is being used for multiple purposes.

Operational requirement tags can be detailed as well, to help incident managers and operations personnel further refine their objectives in response to an incident or event.

Relative links (to the knowledge system base URL) for runbooks and playbooks can be included as tags to assist the responding teams in identifying corresponding process, procedure and documentation.

Table 9 - Use operational tags to inform incident management

Use Case	Tag Key	Rationale	Example Values
Incident Management	`example-inc:incident-management:escalationlog`	The system in use by the supporting team to log incidents	`jira`, `servicenow`, `zendesk`
Incident Management	`example-inc:incident-management:escalationpath`	Path of escalation	`ops-center`, `dev-ops`, `app-team`
Cost Allocation and Incident Management	`example-inc:cost-allocation:CostCenter`	Monitor costs by cost center. This is an example of a dual use tag where the cost center is being used as an application code for incident logging	`123-*`
Backup Schedule	`example-inc:backup:schedule`	Backup schedule of the resource	`Daily`

Use Case	Tag Key	Rationale	Example Values
Playbook / Incident Management	`example-inc:incident-management:playbook`	Documented playbook	`webapp/incident/playbook`

Patching

Organizations can automate their patching strategy for mutable compute environments and keep mutable instances in-line with the defined patch baseline of that application environment by using AWS Systems Manager Patch Manager and AWS Lambda. A tagging strategy for mutable instances within these environments can be managed by assigning said instances to **Patch Groups** and **Maintenance Windows**. See the following examples for a Dev → Test → Prod split. AWS prescriptive guidance is available for the patch management of mutable instances.

Table 10 - Operational tags can be environment specific

Development	Staging	Production
`{ "Tags": [{ "Key": "Maintenance Window", "ResourceId": "i-012345678ab9ab111", "ResourceType": "instance", "Value": "cron(30 23 ? * TUE#1 *)" }, { "Key": "Name", "ResourceId": "i-012345678ab9ab222", "ResourceType": "instance", "Value": "WEBAPP" }, { "Key": "Patch Group", "ResourceId": "i-012345678ab9ab333", "ResourceType": "instance", "Value": "WEBAPP-DEV-AL2" }] }`	`{ "Tags": [{ "Key": "Maintenance Window", "ResourceId": "i-012345678ab9ab444", "ResourceType": "instance", "Value": "cron(30 23 ? * TUE#2 *)" }, { "Key": "Name", "ResourceId": "i-012345678ab9ab555", "ResourceType": "instance", "Value": "WEBAPP" }, { "Key": "Patch Group", "ResourceId": "i-012345678ab9ab666", "ResourceType": "instance", "Value": "WEBAPP-TEST-AL2" }] }`	`{ "Tags": [{ "Key": "Maintenance Window", "ResourceId": "i-012345678ab9ab777", "ResourceType": "instance", "Value": "cron(30 23 ? * TUE#3 *)" }, { "Key": "Name", "ResourceId": "i-012345678ab9ab888", "ResourceType": "instance", "Value": "WEBAPP" }, { "Key": "Patch Group", "ResourceId": "i-012345678ab9ab999", "ResourceType": "instance", "Value": "WEBAPP-PROD-AL2" }] }`

Zero-day vulnerabilities can also be managed by having tags defined to complement your patching strategy. See Avoid zero-day vulnerabilities with same-day security patching using AWS Systems Manager for detailed guidance.

Operational observability

Observability is required to gain actionable insights into the performance of your environments and help you to detect and investigate problems. It also has a secondary purpose that allows you to define and measure key performance indicators (KPIs) and service level objectives (SLOs) such as uptime. For most

organizations, important operations KPIs are mean time to detect (MTTD) and mean time to recover (MTTR) from an incident.

Throughout observability, context is important, as data is collected and then associated tags are gathered. Regardless of the service, application, or application tier that you are focusing on, you can filter and analyze for that specific dataset. Tags can be used to automate on-boarding to CloudWatch alarms so that the right teams can be alerted when certain metric thresholds are breached. For example, a tag key `example-inc:ops:alarm-tag` and the value on could indicate creation of the CloudWatch Alarm. A solution demonstrating this is described in Use tags to create and maintain Amazon CloudWatch alarms for Amazon EC2 instances.

Having too many alarms configured can easily create an alert storm—this is when a large number of alarms or notifications rapidly overwhelm operators and reduce their overall effectiveness, while operators are manually triaging and prioritizing individual alarms. Additional context for the alarms can be provided in the form of tags, which means that rules can be defined within Amazon EventBridge to help ensure that focus is given to the upstream issue rather than downstream dependencies.

The role of operations alongside DevOps is often overlooked, but for many organizations, central operations teams still provide a critical first response outside of normal business hours, (More details can be found about this model in the Operational Excellence whitepaper.) Unlike the DevOps team that owns the workload, they typically do not have the same depth of knowledge, so the context that tags provide within dashboards and alerts, can direct them to the correct runbook for the issue, or initiate an automated runbook (refer to the blog post Automating Amazon CloudWatch Alarms with AWS SageMaker).

Tags for data security, risk management, and access control

Organizations have varying needs and obligations to meet regarding the appropriate handling of data storage and processing. Data classification is an important precursor for several use cases, such as access control, data retention, data analysis, and compliance.

Data security and risk management

Within an AWS environment it's likely you will have accounts with differing compliance and security requirements from a developer sandbox through to an account hosting the production environment for a highly regulated workload, perhaps processing payments. By isolating them into different account you have create the ability to apply distinct security controls, constrain access to sensitive data and reduce the scope of audit for regulated workloads.

Adopting a single standard for all workloads can lead to challenges, whilst many controls apply equally across an environment there are likely to be a few that are excessive or irrelevant for accounts without the need to meet specific regulatory frameworks and those where no personal identifiable data will ever be present, these might include developer sandbox, or workload development accounts. Typically, these will lead to false positive security findings that need to be triaged and closed with no action, this takes effort away from findings that need to be investigated.

Table 11 – Example data security and risk management tags

Use Case	Tag Key	Rationale	Example Values
Incident Management	`example-inc:incident-management:escalationlog`	The system in use by the supporting team to log incidents	`jira`, `servicenow`, `zendesk`

Use Case	Tag Key	Rationale	Example Values
Incident Management	`example-inc:incident-management:escalationpath`	Path of escalation	`ops-center`, `dev-ops`, `app-team`
Data Classification	`example-inc:data:classification`	Classify data for compliance and governance	`Public`, `Private`, `Confidential`, `Restricted`
Compliance	`example-inc:compliance:framework`	Identifies the compliance framework the workload is subject to	`PCI-DSS`, `HIPAA`

Manually managing different controls across an AWS environment is both time consuming and error prone. The next step is to automate the deployment of appropriate security controls, and configure inspection of resources, based on the classification of that account. By applying tags to the accounts and the resources within them the deployment of controls can be automated and configured appropriately for the workload.

Working through an example:

A workload includes an S3 bucket, with the tag `example-inc:data:classification` with the value `Private`. The security tooling automation, deploys: AWS Config rule `s3-bucket-public-read-prohibited` which checks the Amazon S3 bucket's Block Public Access settings, the bucket policy, and the bucket access control list (ACL), confirming the bucket's configuration is appropriate for its data classification. To ensure the content of the bucket is consistent with the classification Amazon Macie can be configured to check for personal identifiable information (PII). The blog Using Amazon Macie to Validate S3 Bucket Data Classification, explores this pattern in greater depth.

Certain regulatory environments, such as insurance and healthcare, might be subject to mandatory data retention policies. Data retention using tags can be an effective and simple way, combined with Amazon S3 Lifecycle policies, to scope object transitions to a different storage tier. Also, Amazon S3 Lifecycle rules can be used to expire objects for data deletion after the mandatory hold period expires. See Simplify your data lifecycle by using object tags with Amazon S3 Lifecycle for an in-depth guide to this process.

Additionally, when triaging or addressing security finding, tags can provide the investigator important context that help qualify the risk and aid in engaging the appropriate teams to investigate or mitigate the finding.

Tags for identity management and access control

When managing access control across an AWS environment with AWS IAM Identity Center (successor to AWS Single Sign-On), tags can enable several patterns for scaling. There are several delegation patterns that can be applied, some are based on tagging. We'll address them individually and provide links to further reading on each.

Delegated management of identity within IAM Identity Center

The pattern provides the means to delegate the right to assign users and groups with permissions for groups of accounts within IAM Identity Center, based on tags on the resources within IAM Identity Center.

This allows for a number of delegation models:

- A team can manage all permission sets for the set of accounts they are responsible for.
- A centralized support team (such as Operations or Security Operations) can apply read-only permission sets to any accounts.
- A team can manage only the permission sets intended for a specific OU, for which they are responsible.

The resources being tagged for this approach are typically long lived, exist within IAM Identity Center, and are hosted in the management account. Typically, this activity will be owned by team with a focus on security and identity. This makes that one team responsible for ensuring the accuracy and consistency of the tagging.

This approach offers considerable flexibility and the ability to delegate management control of identity within an organization. However, this approach may not be appropriate for some organizations, such as those in highly regulated industries. The How to delegate management of identity blog post explores how to implement these delegation models in detail.

ABAC for individual resources

IAM Identity Center users and IAM roles support attribute-based access control (ABAC), which allows you to define access to operations and resources based on tags. ABAC helps reduce the need to update permission policies and helps you to base access off of employee attributes from your corporate directory. If you are already using a multi-account strategy, ABAC can be used in addition to role-based access control (RBAC) to provide multiple teams operating in the same account granular access to different resources. For example, IAM Identity Center users or IAM roles can include conditions to limit access to specific EC2 instances which otherwise would have to be explicitly listed in each policy in order to access them.

Since an ABAC authorization model depends on tags for access to operations and resources, it is important to provide guardrails to prevent unintended access. SCPs can be used to protect tags across your organization by only allowing tags to be modified under certain conditions. The blogs Securing resource tags used for authorization using a service control policy in AWS Organizations and Permissions boundaries for IAM entities provide information on how to implement this.

Where long-lived EC2 instances are being used to support more traditional operations practices then this approach can be utilized, the blog Configure IAM Identity Center ABAC for EC2 instances and Systems Manager Session Manager discusses this form of attribute based access control in more detail. As mentioned earlier, not all resource types support tagging, and of those that do, not all support enforcement using tag policies, so it's a good idea to evaluate this prior to starting to implement this strategy on an AWS account.

To learn about services that support ABAC, see AWS services that work with IAM.

Conclusion

AWS resources can be tagged for a variety of purposes, from implementing a cost allocation strategy to supporting automation or authorizing access to AWS resources. Implementing a tagging strategy can be challenging for some organizations, owing to the number of stakeholder groups involved and considerations such as data sourcing and tag governance.

In this whitepaper, we've outlined recommendations regarding designing and implementing a tagging strategy in an organization based on operational practices, defined use cases, stakeholders involved in the process, and tools and services provided by AWS. When it comes to a tagging strategy, it's a process of iteration and improvement, where you start small from your immediate priority, identify relevant use cases across your organization, and then implement and grow the tagging schema as you need to, while continuously measuring and improving effectiveness. We've pointed out that a well-defined set of tags within your organization will allow you to relate AWS usage and consumption to teams responsible for the resources and business purpose for which they exist, in order to align with organizational strategy and value.

Contributors

Contributors to this document include:

- Chris Pates, Sr Specialist Technical Account Manager, Amazon Web Services
- Vijay Shekhar Rao, Enterprise Support Lead, Amazon Web Services
- Nataliya Godunok, Sr Specialist Technical Account Manager, Amazon Web Services
- Yogish Kutkunje Pai, Sr Solutions Architect, Amazon Internet Services Private Limited
- Jamie Ibbs, Sr Specialist Technical Account Manager, Amazon Web Services

Further reading

For more information, refer to

- AWS re:Invent 2020: Working backwards: Amazon's approach to innovation
- AWS Prescriptive Guidance: Automated patching for mutable instances in the hybrid cloud using AWS Systems Manager
- AWS Architecture Center

AWS Well-Architected

- AWS Well-Architected Framework
- Operational Excellence Pillar - AWS Well-Architected Framework
- Plan for Disaster Recovery (DR) - AWS Well-Architected Reliability pillar
- Cost Optimization Pillar - AWS Well-Architected Framework
- AWS Well-Architected Labs: Enable AWS-Generated Cost Allocation Tags
- AWS Well-Architected Labs: Tag Policies
- AWS Well-Architected Labs: AWS CUR Query Library

AWS blogs

- AWS Health Aware – Customize AWS Health Alerts for Organizational and Personal AWS Accounts
- Automatically tag new AWS resources based on identity or role
- How to Automatically Tag Amazon EC2 Resources in Response to API Events
- AWS Generated vs. User-Defined Cost Allocation Tag
- Cost Tagging and Reporting with AWS Organizations
- Patching your Windows EC2 instances using AWS Systems Manager Patch Manager
- Avoid zero-day vulnerabilities with same-day security patching using AWS Systems Manager

AWS documentation

- Using Cost Allocation Tags - AWS Billing and Cost Management and Cost Management and Cost Management
- What are AWS Cost and Usage Reports
- AWS Resource Groups API Reference
- How can I use IAM policy tags to restrict how an EC2 instance or EBS volume can be created?
- Mutable vs immutable update models

Other

- Bryar, C. and Carr, B. (2021). *Working Backwards: Insights, Stories, and Secrets from Inside Amazon*. London Macmillan.
- AWS CloudFormation Guard (GitHub)

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Minor revision (p. 33)	Updated reference in ABAC for individual resources.	February 24, 2023
Minor revision (p. 33)	Updated guide to align with the IAM best practices. For more information, see Security best practices in IAM.	February 6, 2023
Major revision (p. 33)	Added more specific reference for resource types supported by AWS Config rule `required_tags`.	January 18, 2023
Major revision (p. 33)	Updated to include the latest practices and service capabilities, especially in the area of identity.	September 29, 2022
Minor update (p. 33)	Fixed table formatting in PDF version.	April 25, 2022
Major revision (p. 33)	Updated document structure and expanded Tagging Strategy and Use Cases sections. Added more prescriptive guidance based on the latest tools, techniques, and available resources.	April 22, 2022
Initial publication (p. 33)	Whitepaper first published.	December 1, 2018

Note
To subscribe to RSS updates, you must have an RSS plugin enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.