

Azure Policy initiatives

1. Azure Policy design principles

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/azure-policy-design-principles>

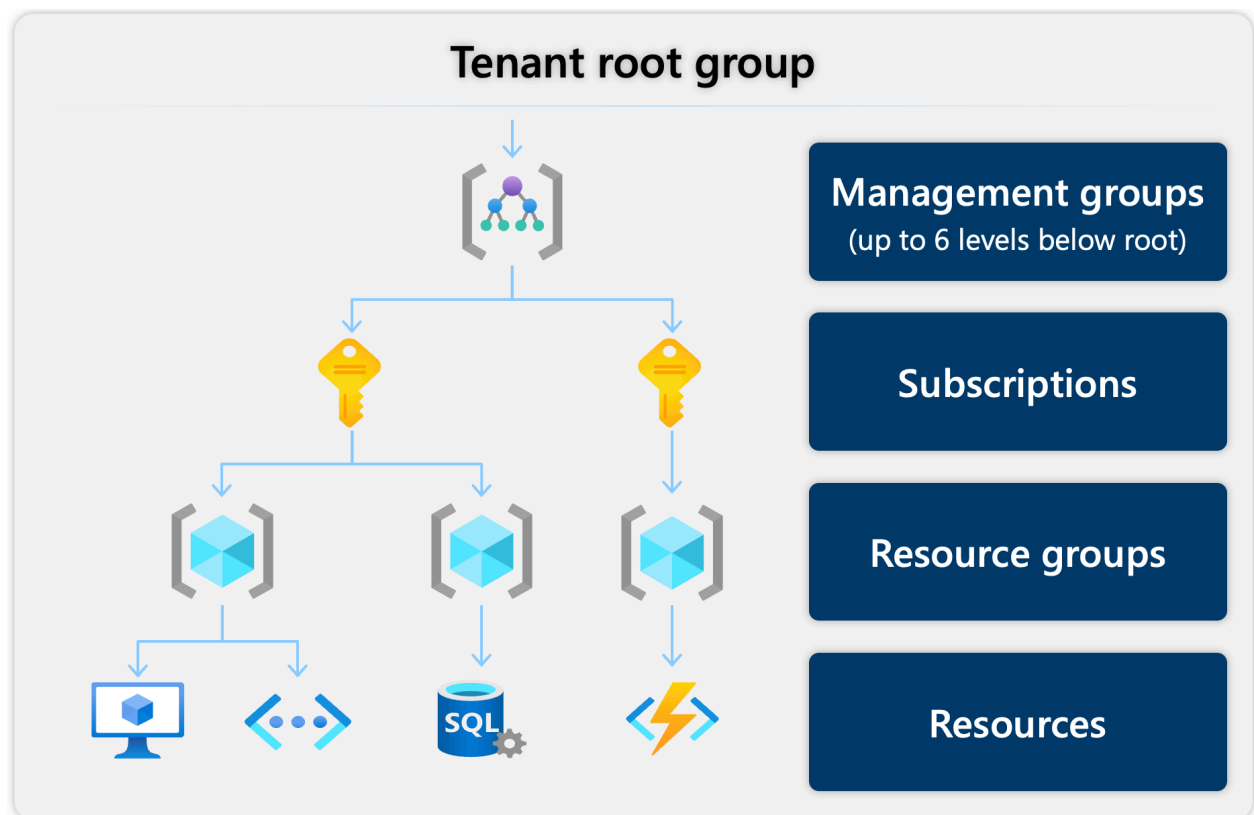
Azure Policy design principles

Completed

Governance provides mechanisms and processes to maintain control over your applications and resources in Azure. It involves planning your policy in Azure Policy and setting strategic priorities. While designing your policy, you must organize your cloud-based resources to secure, manage, and track costs that are related to your workloads.

Hierarchy for governance

Azure provides four levels of management to establish proper governance: Management groups, Subscriptions, Resource groups, and Resources. You can build a flexible structure of management groups and subscriptions to organize your resources into a hierarchy for unified policy and access management. The following diagram shows an example of creating a hierarchy for governance by using management groups.



The Azure structure starts with the tenant root group at the top, followed by a hierarchy of management groups, which can extend to six levels beneath the root. The definition of each level in the management hierarchy and relationship between these levels is as follows:

Concept	Description
Resource	A resource is the basic building block of Azure, and it includes instances of services that you create, provision, deploy, and so on. Virtual machines (VMs), virtual networks, databases, AI services, and so on, are considered resources in Azure.
Resource groups	Resource groups are groupings of resources. When you create a resource, you must place it into a resource group. While a resource group can contain many resources, a single resource can only be in one resource group at a time. When you apply an action to a resource group, that action applies to all resources in the resource group. If you delete a resource group, all resources are deleted. If you grant or deny access to a resource group, all resources in the resource group are also granted or denied access.
Subscriptions	In Azure, subscriptions are a unit of management, billing, and scale. Similar to how resource groups are a way of logically organizing resources, subscriptions allow you to logically organize your resource groups and facilitate billing. Each subscription has limits or quotas on the number of resources that you can create and use. Organizations can use subscriptions to manage costs and the resources that users, teams, and projects create. Using Azure requires an Azure subscription. An Azure subscription provides you with authenticated and authorized access to Azure products and services. It also allows you to

Concept	Description
	provision resources. An Azure subscription links to an Azure account, which is an identity in Microsoft Entra ID or in a directory that Microsoft Entra ID trusts.
Management groups	Azure management groups provide a level of scope above subscriptions. If you have many subscriptions, you might need a way to efficiently manage access, policies, and compliance for those subscriptions. You organize subscriptions into containers called management groups and then apply governance conditions to the management groups. Management groups give you enterprise-grade management at a large scale, no matter what type of subscriptions you might have. Management groups can be nested.

You can apply management settings at any of these levels of scope. The level that you select determines how widely the setting is applied. Lower levels inherit settings from higher levels. For example, when you apply a policy to the subscription, the policy is applied to all resource groups and resources in that subscription. When you apply a policy on the resource group, that policy is applied to that resource group and all its resources. However, another resource group won't have that policy assignment. All subscriptions within a management group automatically inherit the conditions that are applied to the management group.

Introduction to Azure Resource Manager

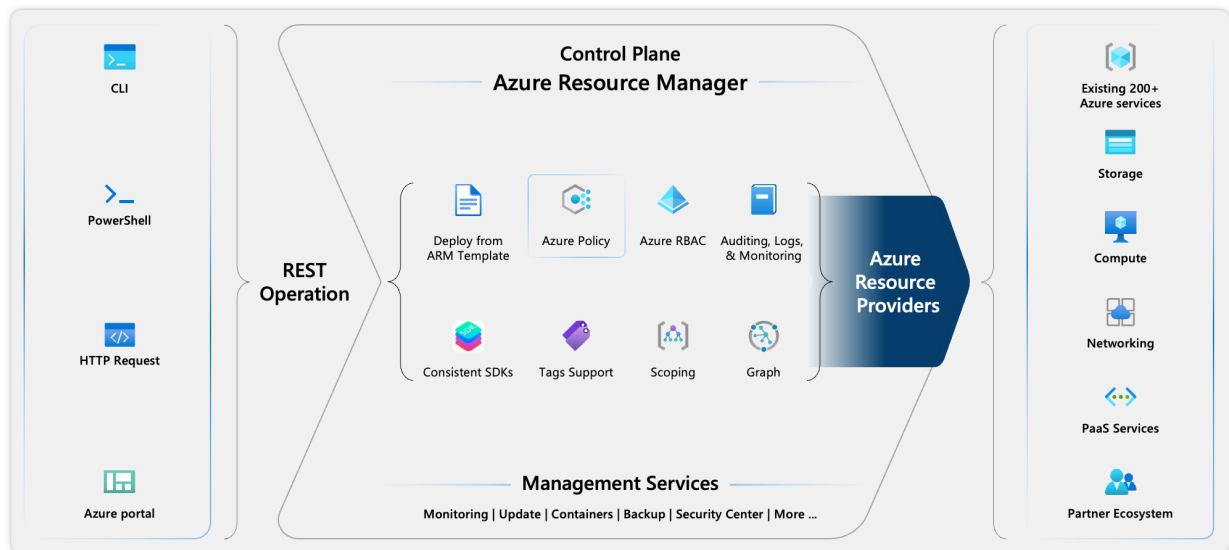
Azure Resource Manager is the deployment and management service for Azure. It provides a management layer that allows you to create, update, and delete resources in your Azure account.

Azure operations are classified into two main types: control plane and data plane. The control plane helps you manage resources in your subscription, while the data plane allows you to access the capabilities provided by instances of specific resource types.

Control plane

Azure Policy operates in the control plane to enforce rules and compliance on your resources. Azure Resource Manager manages all control plane operations in Azure and includes the different components that are centralized between the different services. Azure Policy is integrated with Azure Resource Manager.

Azure Policy and Azure Resource Manager



Azure Resource Manager manages essential functions, such as template-based deployments, role-based access control (RBAC), auditing, monitoring, and tagging, which provides a unified management experience for Azure resources after deployment. For example, consider a scenario where you have a storage account. With Azure Resource Manager, you can create the storage account and enforce a policy that mandates encryption for all storage accounts.

When you send a control plane request through any of the Azure APIs, tools, or SDKs, Azure Resource Manager receives the request. All capabilities that are available in the portal are also available through PowerShell, Azure CLI, REST APIs, and client SDKs. These tools use the same API to process requests, ensuring uniform results and capabilities. The Resource Manager authenticates and authorizes the request before forwarding it to the appropriate Azure resource provider that completes the operation.

Data plane

The data plane is where the actual data operations occur, and Azure Policy ensures that the resources you interact with in the data plane are compliant with your policies. Data plane operations involve direct interaction with the data stored in a resource. Continuing with the previous example, you engage with the storage account to upload or download files. This interaction is handled directly by the data plane of the storage account rather than being managed by Azure Resource Manager.

Data plane operations aren't limited to REST API. Requests are made directly to the data endpoints of services (for example, accessing data in Microsoft Azure Storage, querying an SQL database, or reading secrets from Microsoft Azure Key Vault). Each Azure service handles these requests internally, bypassing Azure Resource Manager, and directly managing the data through its resource provider. Service-specific access controls, such as RBAC or access control lists (ACLs), often manage data plane permissions. The service responds with the data or result of the data operation, ensuring that the requester has the correct permissions.

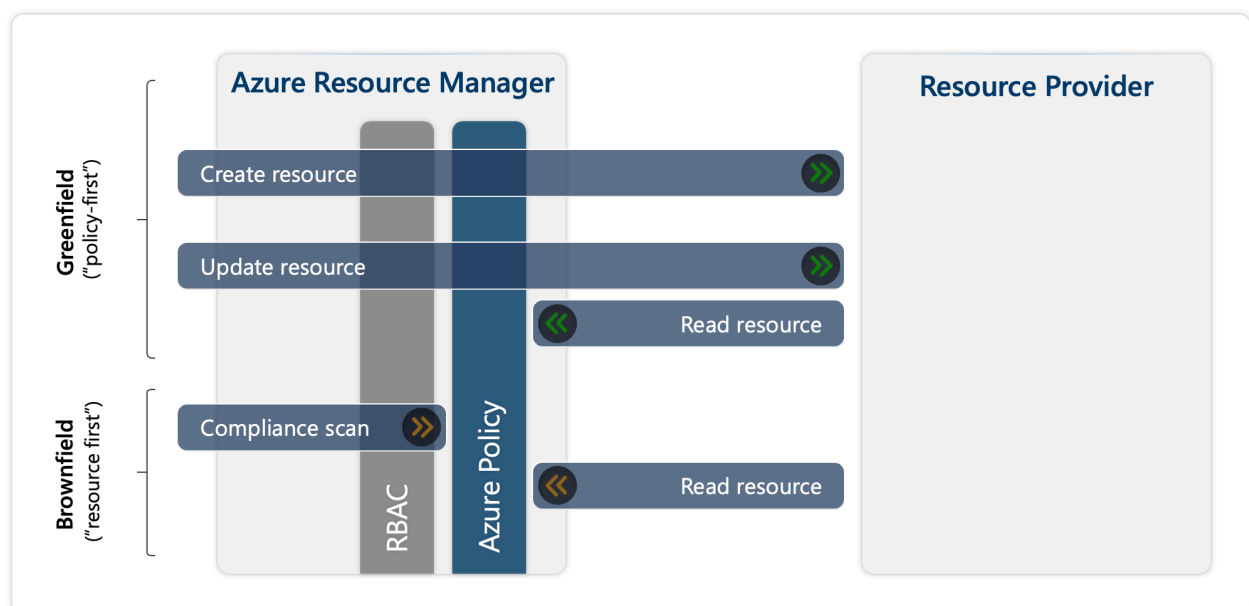
Azure Policy allows individual Azure services to implement an Azure Policy extension, enhancing policy behavior and integration with specific resource providers. Azure Policy currently supports data plane operations through the following resource provider modes:

- **Microsoft.Kubernetes.Data** - Used for managing Kubernetes clusters and components such as pods, containers, and ingresses.
- **Microsoft.KeyVault.Data** - Used for managing vaults and certificates in Azure Key Vault.
- **Microsoft.Network.Data** - Used for managing Microsoft Azure Virtual Network Manager custom membership policies by using Azure Policy.
- **Microsoft.ManagedHSM.Data** - Used for managing Azure Key Vault Managed HSM keys by using Azure Policy.
- **Microsoft.DataFactory.Data** - Used for using Azure Policy to deny Microsoft Azure Data Factory outbound traffic domain names.
- **Microsoft.MachineLearningServices.v2.Data** - Used for managing Microsoft Azure Machine Learning model deployments. This Resource Provider mode reports compliance for newly created and updated components.

For more information, see [Resource Provider Modes](#).

Operation flows of Azure Resource Manager

Azure Resource Manager includes two scenarios for handling Azure requests: **Greenfield** and **Brownfield**. As you deploy resources, Azure Resource Manager understands when to create new resources and when to update existing resources.



Greenfield refers to a scenario where an Azure Policy (policy-first) exists, and when you're creating or updating an Azure resource.

For example, you're creating a new resource by calling an HTTPS REST API into Azure Resource Manager, that is, targeting a specific resource provider. In Azure Resource Manager, the request goes

through different layers, including role-based access control (RBAC) and Azure Policy. These layers are only two of the many other layers, but it's important to understand that Azure Policy comes after RBAC. If you don't have permissions to run a given operation, then Azure Policy isn't considered or called because it would fail on the RBAC stage. If you have permissions, the request goes through Azure Policy and is evaluated against any policy. For resource updates, you send only the changes (delta) in the request payload in the existing resource. Azure Policy requires full visibility of the current state of the resource. Azure Policy merges the delta that you're sending by reading the current state. Then, the target state is obtained as a result of the merger, and that's what gets evaluated against your policies.

Brownfield is the scenario where the resources exist already (resource-first), and you're assigning a new Azure Policy to those resources.

In this case, policy evaluation happens through a compliance scan, which runs automatically every 24 hours, or it can be manually triggered. The duration of the scan is unpredictable, but after it's completed, the compliance state of existing resources is updated. To perform this evaluation, Azure Policy reads all existing resources in the scope. You can create an Azure policy that prohibits resources from being created outside of a certain region, such as West Europe. Existing resources outside this region aren't deleted but are flagged as noncompliant after the policy is applied, and future attempts to create resources outside West Europe fail.

2. Summary

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/summary>

Summary

Completed

Azure Policy is a crucial component of the governance model in the Cloud Adoption Framework for Azure, which is designed to balance control and stability with speed and results. It helps you enforce organizational and regulatory standards and assess compliance at scale through built-in and custom policies and policy initiatives.

The module covered the hierarchical organization of Azure resources, policy operations in Greenfield and Brownfield scenarios, and the various components of policy definitions. You also delved into the evaluation and effects of policies, safe deployment practices, and integration with Event Grid for automated actions based on policy state changes. Key points included:

- Importance of careful policy design
- Testing to ensure effective governance without disrupting operations
- Logical operators and conditions in policy evaluation

- Supported effect types such as *disabled*, *modify*, *deny*, *audit*, *deployIfNotExists*, and *manual*

Additionally, the module emphasized starting with *enforcementMode* deactivated for new policies to test their impact and then deploying policies in rings to gradually expand to production environments.

For more information, see the [Azure Policy](#) documentation.

3. Evaluation of resources through Azure Policy

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/azure-policies-evaluation-resources>

Evaluation of resources through Azure Policy

Completed

A significant benefit of Azure Policy is the insight and controls that it provides over resources in a subscription or management group of subscriptions. This control can be used to prevent resources from being created in the wrong location, enforce common and consistent tag usage, or audit existing resources for appropriate configurations and settings. Before you review compliance data and react to it, you need to understand the evaluation triggers, timings, and the resource compliance states.

Evaluation triggers

Evaluations of assigned policies and initiatives happen as the result of various events:

- A policy or initiative is newly assigned to a scope
- A policy or initiative already assigned to a scope is updated
- A resource is deployed to or updated in a scope with an assignment through Azure Resource Manager, REST API, or a supported SDK
- A subscription (resource type Microsoft.Resources/subscriptions) is created or moved in a management group hierarchy with an assigned policy definition that targets the subscription resource type
- A policy exemption is created, updated, or deleted
- Standard compliance evaluation cycle
- The machine configuration resource provider is updated with compliance details by a managed resource
- On-demand scan

For more information, see [Evaluation triggers](#).

Evaluation timing

When you're working with policy assignments in Azure, you need to understand the behavior and timing of compliance scans, especially in Brownfield scenarios, where new policies are applied to existing resources. Compliance scans through Azure policies are triggered by various methods:

- **Automatic full scan** - A full compliance scan is triggered automatically every 24 hours.
- **Manual scan for Brownfield scenarios** - In cases where a new policy is applied to existing resources (Brownfield scenarios), you can manually trigger a compliance scan by running *az policy state trigger-scan*.

When you assign a new policy, a delay can occur in the policy taking effect, which can be up to 30 minutes. The Azure Resource Manager cache holds session data, and it can take time for the policy to propagate in the same session. To bypass the caching delay, you can sign out and sign back in to refresh the Azure Resource Manager cache, which ensures that the new policy is applied immediately to the defined scope.

After the scan starts, several factors influence how long it takes for a compliance scan to complete:

- **Policy definitions** - The size and complexity of the policy definitions can increase scan time.
- **Number of policies** - The more policies applied, the longer the scan might take.
- **Scope size** - The size of the resource scope assigned to the policy also plays a role.
- **System load** - Compliance scans are a low-priority operation, meaning that if the system is busy with more critical tasks, the scan might take longer. The system prioritizes interactive and high-importance operations, so scans might take several minutes, or tens of minutes, even in smaller environments.
- **Synchronous scan (Low-Priority Execution)** - Because compliance scans are synchronous and assigned a low priority in Azure's system, they're delayed if the system is busy. This scan can significantly extend the time it takes for the scan to complete, even for smaller scopes or policies.

This understanding of the compliance scan process and potential delays allows you to better manage the application and avoid unnecessary waiting, especially in environments with complex or extensive policy definitions.

Resource compliance states

When initiative or policy definitions are assigned, Azure Policy determines which resources are applicable. Then, it evaluates those resources that aren't excluded or exempted. Evaluation provides one of the compliance states to each resource based on conditions in the policy rule and each resource's adherence to those requirements.

- **Non-compliant**
- **Compliant**
- **Error** (for template or evaluation error)

- **Conflicting** (two or more policy assignments in the same scope with contradicting rules, such as two policies appending the same tag with different values)
- **Protected** (resource covered under an assignment with a *denyAction* effect)
- **Exempted Unknown** (default state for definitions with a *manual* effect)

When multiple resources or policies have varying compliance states, the overall compliance state is assessed individually for each resource and for each policy assignment. Azure Policy ranks each compliance state so that one wins over another in this situation. The rank order of the states is as given in the previous list of compliance states.

The compliance percentage is determined by dividing **Compliant**, **Exempt**, and **Unknown** resources by total resources. Total resources include resources with **Compliant**, **Non-compliant**, **Unknown**, **Exempt**, **Conflicting**, and **Error** states.

For more information on when the policies return these states for any particular resource, see [Azure Policy compliance states](#).

Enforcement Mode

enforcementMode is a property of a policy assignment that lets you deactivate the enforcement of certain policy effects. This mode allows you to test the policy's outcome on existing resources without initiating the policy effect or triggering entries in the [Azure Activity log](#). The *enforcementMode* can be changed to Enabled after the policy is thoroughly tested.

This scenario is commonly referred to as *What If* and aligns to safe deployment practices. The *enforcementMode* is different from the *disabled* effect. The *disabled* effect prevents resource evaluation from happening at all while *enforcementMode* lets the evaluation happen without the effect taking place.

The following table describes this property's values.

Mode	JSON value	Type	Remediate manually	Activity log entry	Description
Enabled	Default	string	Yes	Yes	The policy effect is enforced during resource creation or update.
Disabled	DoNotEnforce	string	Yes	No	The policy effect isn't enforced during resource creation or update.

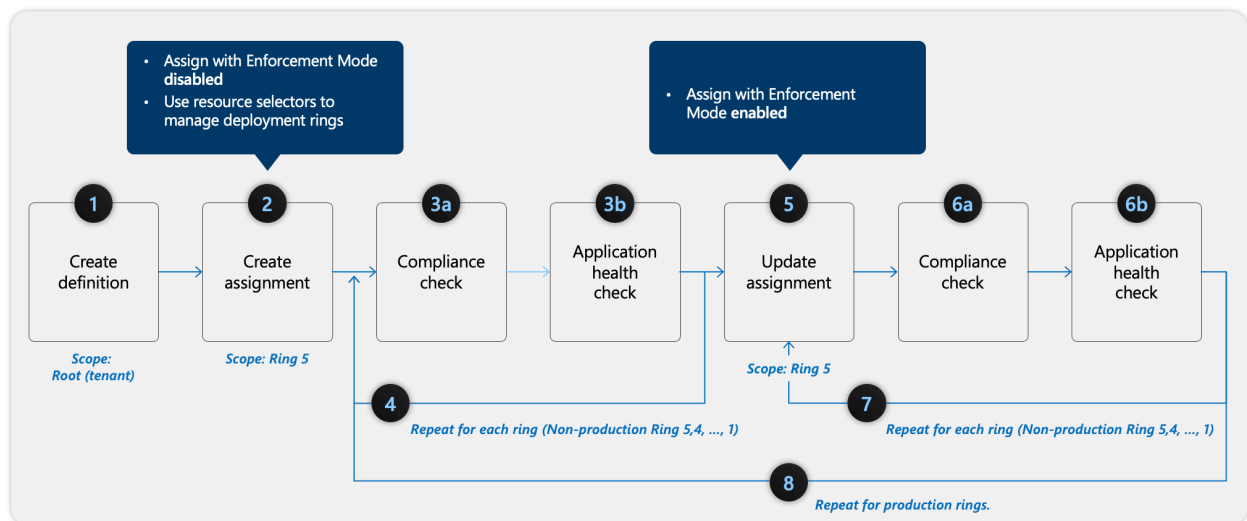
If *enforcementMode* isn't specified in a policy or initiative definition, the value *Default* is used. Remediation tasks can be started for *deployIfNotExists* policies, even when *enforcementMode* is set to *DoNotEnforce*.

Policy enforcement and safe deployment best practices

Without the appropriate knowledge of best practices and proper testing, applying a set of policy to an existing environment that's running production workloads can result in unintended behaviors of policy resources. Treating policy as code (keeping your policy definitions in source control, and whenever a change is made, testing and validating that change) allows you to automate testing and make sure that no manual error factor happens. The best practices framework focuses on minimizing the impact of policy changes while ensuring compliance, and it includes two aspects:

- **First aspect** - Start from Assignments of new policies with *enforcementMode* Disabled. When assigning policies that include deny or modify actions, beginning with *enforcementMode* Disabled allows you to view the compliance state and evaluate policy outcomes without triggering actions or denying operations. This "what-if" scenario minimizes impact and helps identify issues in the new policies or changes without disrupting the environment.
- **Second aspect** - Deploy policies in deployment rings. To control potential negative impacts, policies should be deployed gradually in smaller subsets and then in bigger sets. You can start with test and development environments and then move to production by applying the policy to a small subset first. This strategy helps in testing the policy thoroughly. Gradually expanding the scope (through deployment rings) can cover the full production environment.

The following diagram shows how to apply the safe deployment best practices framework for Azure Policy assignments.



The following steps correspond with the outlined steps in the preceding screenshot:

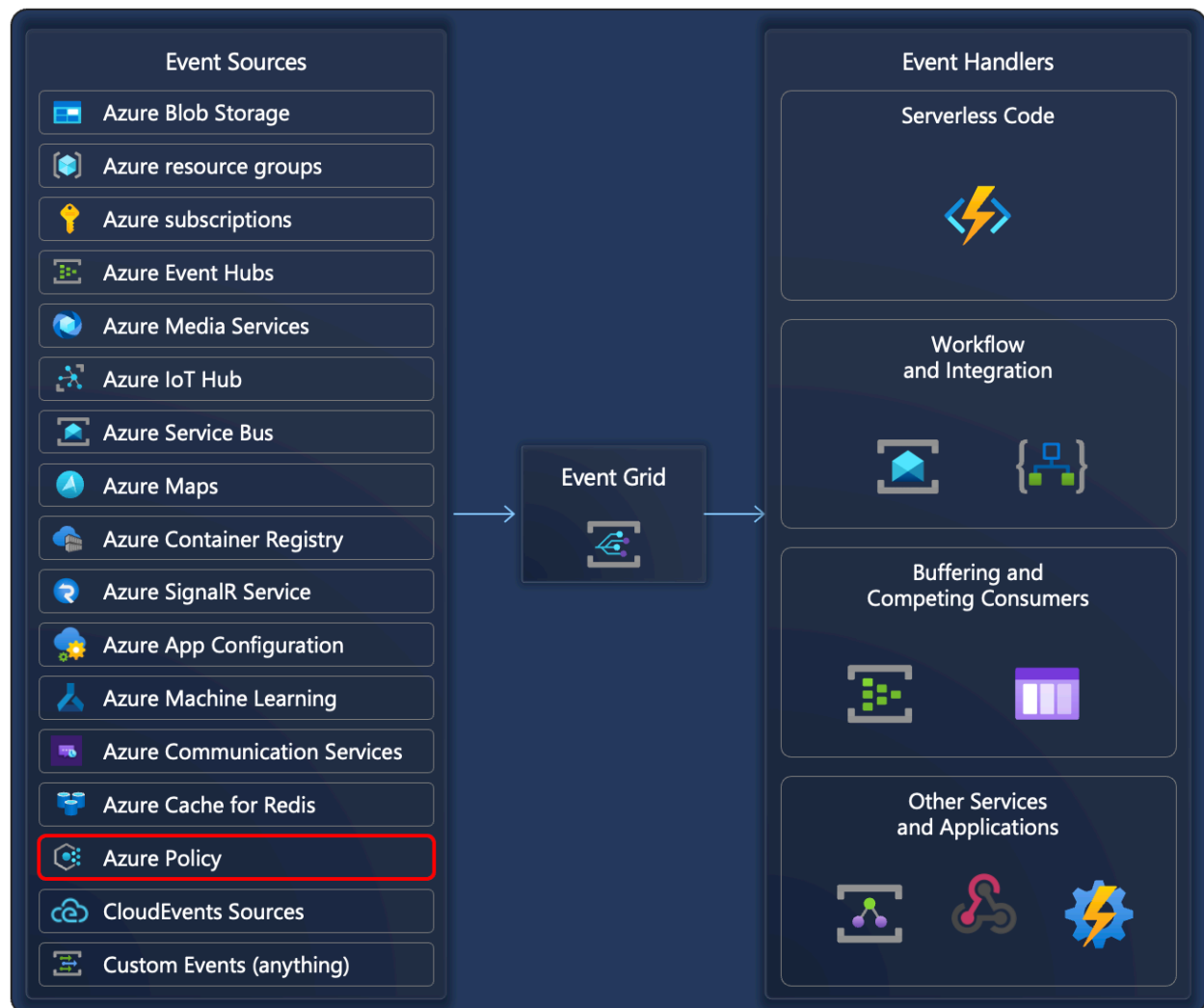
1. **Create definition** - Begin by defining the policy definition with the scope as root (tenant).
2. **Create assignment** - Define deployment rings (1 to 5) by using resource selectors. Assign the policy to a specific scope (such as a resource group, subscription, or management group) in Ring 5. Assign with *enforcementMode* Disabled to evaluate compliance without enforcing changes.
3. a) **Compliance check** - Verify that the policy is being applied correctly and that the desired compliance state is achieved for the resources in Ring 5.

- b) **Application health check** - Assess the impact of the policy for the resources in Ring 5. Ensure that no unexpected side effects exist.
4. **Repeat for each ring (Non-production)** - Repeat step 3 for all nonproduction environment rings.
5. **Update assignment (Optional)** - If necessary, adjust the policy definition or assignment based on the evaluation of resources of the nonproduction environment and then reassign it to the resources in Ring 5 with the *enforcementMode* Enabled.
6. a) **Compliance check** - Reevaluate compliance after making changes (same as step 3a).
- b) **Application health check** - Again, verify that the policy isn't causing issues (same as step 3b).
7. **Repeat for each ring (Non-production)** - Repeat step 6 for all nonproduction environment rings.
8. **Repeat for production rings** - After the policy is validated in a nonproduction environment, gradually deploy it to production environments, starting with a smaller subset (ring) and expanding the scope over time.

For more detailed steps on the safe deployment of Azure Policy assignments with different effects, see [Safe deployment of Azure Policy assignments](#).

Reacting to policy state changes

Azure Policy events allow applications to react to state changes. This integration is done without the need for complicated code or expensive and inefficient polling services. Events from Azure Policy (Event Source) are pushed through Microsoft Azure Event Grid to Event Handlers.



Azure Event Grid provides reliable delivery services to your applications through rich retry policies and dead-letter delivery. Event Grid takes care of the proper routing, filtering, and multicasting of the events to destinations through Event Grid subscriptions. For more information, see [Azure Event Grid](#).

An Event Handler is the place where the event is sent. Several services, such as Microsoft Azure Functions, Microsoft Azure Logic Apps, or your own custom HTTP listener, can be configured to handle events. You can also use any webhook for handling events.

For more information, see [Event handlers in Azure Event Grid](#), [Reacting to Azure Policy state change events](#), and the [Route policy state change events to Event Grid with Azure CLI](#) tutorial.

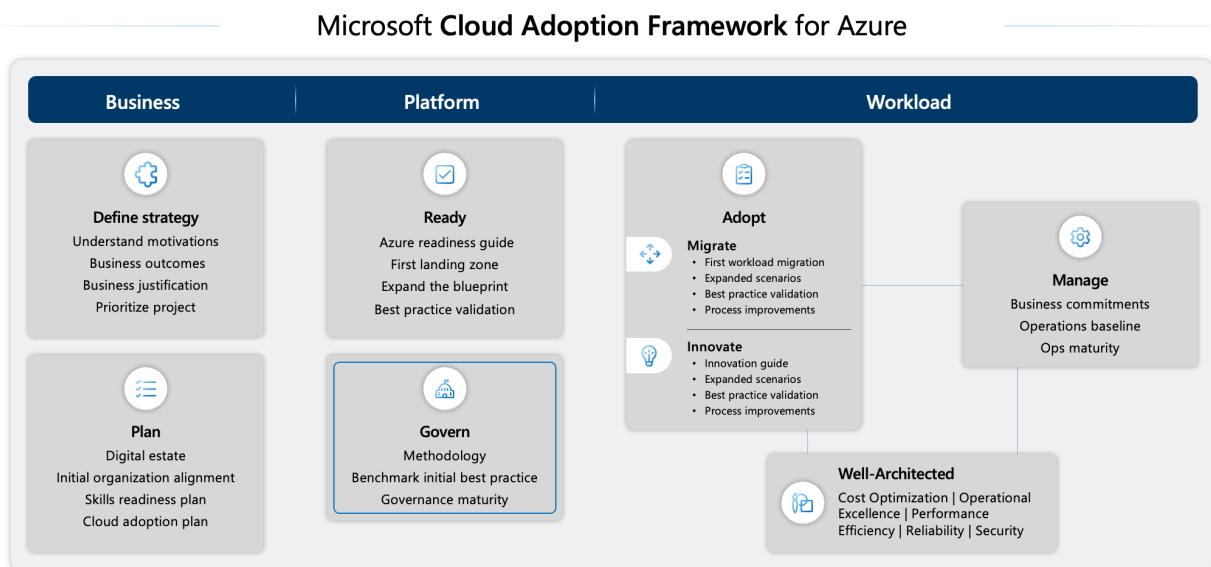
4. Cloud Adoption Framework for Azure

Cloud Adoption Framework for Azure

Completed

The Microsoft Cloud Adoption Framework for Azure offers comprehensive technical guidance for Microsoft Azure. This end-to-end framework helps cloud architects, IT experts, and business leaders reach their cloud adoption objectives. Cloud Adoption Framework includes best practices, documentation, and tools that Microsoft employees, partners, and customers contribute to formulate and implement effective business and technology strategies for the cloud. For more information, see [Microsoft Cloud Adoption Framework for Azure documentation - Cloud Adoption Framework | Microsoft Learn](#).

The following diagram provides high-level information about different methodologies that are included in Microsoft Cloud Adoption Framework for Azure for each phase of your cloud adoption life cycle. Within the framework, Microsoft Azure Policy plays a significant role in the governance framework to help govern your cloud environment and workloads.



Cloud governance refers to the management of cloud usage in your organization. The Cloud Adoption Framework - Govern methodology offers a systematic framework for setting up and improving cloud governance in Azure. This guidance applies to organizations across various industries and it addresses crucial areas, such as regulatory compliance, security, operations, cost management, data, resource management, and AI. It's essential for defining and maintaining efficient cloud use.

Comprehensive cloud governance oversees all aspects of cloud use, minimizes various risks (such as compliance, security, resource management, and data-related concerns), and optimizes cloud

operations throughout the organization. It ensures that cloud activities are consistent with the overall cloud strategy, facilitating the achievement of business objectives with reduced obstacles.

Steps for cloud governance

Cloud governance is a continuous process. It requires ongoing monitoring, evaluation, and adjustments to adapt to evolving technologies, risks, and compliance requirements. The Cloud Adoption Framework - Govern methodology divides cloud governance into five steps.



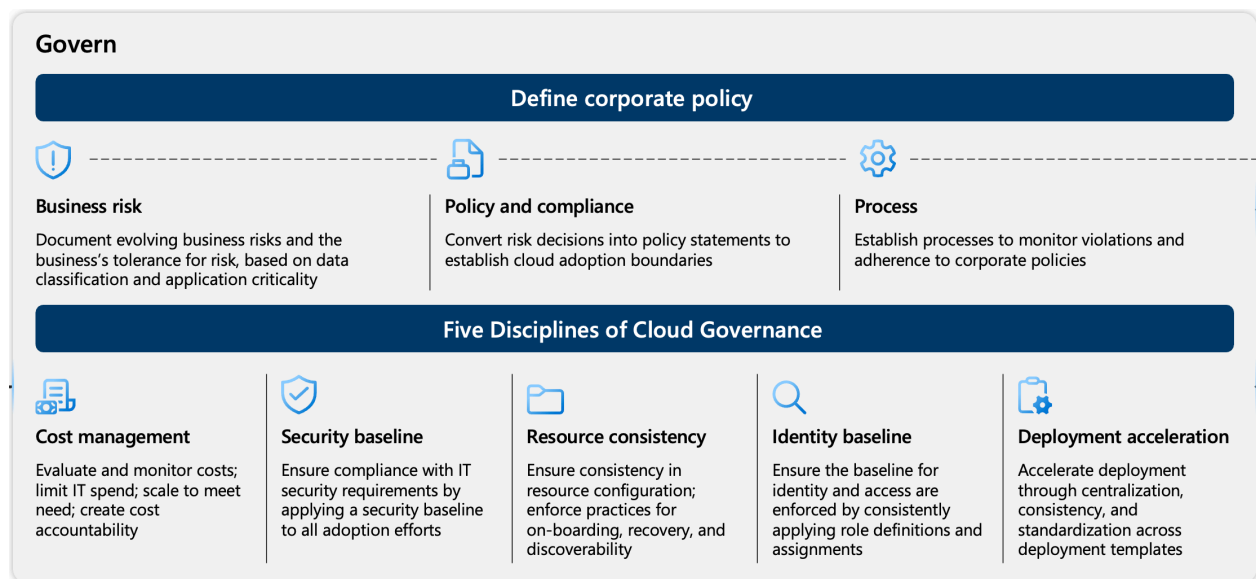
1. **Build a governance team** - Establish a dedicated cloud governance team that's responsible for defining, maintaining, and reporting on the progress of cloud governance policies.
2. **Assess cloud risks** - Conduct a thorough risk assessment that's unique to your organization, addressing all risk categories, including regulatory compliance, security, operations, costs, data management, resource management, and AI-related risks.
3. **Document cloud governance policies** - Clearly document cloud governance policies that dictate acceptable cloud usage and outline the rules and guidelines that mitigate identified risks.
4. **Enforce cloud governance policies** - Implement a systematic approach to ensure compliance with cloud governance policies. Use automated tools alongside manual oversight to enforce compliance. These tools help set guardrails, monitor configurations, and ensure adherence to policies.
5. **Monitor cloud governance** - Regularly monitor cloud usage and the governance teams to ensure ongoing compliance with the established cloud governance policies.

You should complete all five steps to establish cloud governance and regularly iterate on steps 2-5 to maintain cloud governance over time.

Considerations for defining a cloud governance policy

The key considerations when defining a corporate cloud governance policy are as follows:

- **Business risk** – You must document the evolving business risks and the business's tolerance for risk based on data classification and application criticality.
- **Policy and compliance** – You must convert risk decisions into policy statements to establish cloud adoption boundaries efficiently.
- **Process** – You must establish processes to monitor violations and adherence to corporate policies.



The five core disciplines of cloud governance are as follows:

- **Cost management** – Evaluates and monitors costs, including controlling IT expenditures to establish well-defined cost management. It also includes adjusting resources according to demand. It's crucial to exercise control over cloud expenditure to derive greater value from your investments.
- **Security baseline** – Ensures compliance with IT security requirements by applying a security baseline to all adoption efforts.
- **Resource consistency** – Ensures consistency in resource configuration and enforcing practices for onboarding, recovery, and discoverability.
- **Identity baseline** – Ensures that the baseline for identity and access is enforced by consistently applying role definitions and assignments.
- **Deployment acceleration** – Accelerates the deployment of policies through centralization, consistency, and standardization across deployment templates.

Adopting these measures simplifies the compliance process and also enhances the security and efficiency of your cloud environment.

Cloud governance with Azure Policy

Azure's primary governance tool is [Azure Policy](#). Azure Policy facilitates the governance of all resources, including current and forthcoming resources. It helps to enforce organizational standards and to assess compliance at scale by establishing guardrails across various resources.

Azure Policy allows for centralized management, allowing you to track compliance status and investigate changes leading to noncompliance. You can consolidate all compliance data into a singular repository, which simplifies auditing processes for effective cloud compliance and resource governance. The compliance dashboard offered by Azure Policy presents an aggregated view of the environment's overall state, with the ability to examine details at each resource and each policy level.

Azure Policy ensures consistent adherence to compliance standards and prevents misconfigurations. It also helps bring your resources to compliance through bulk remediation for existing resources and automatic remediation for new resources. Additionally, embedding Azure Policy directly into the Azure platform can significantly reduce the need for external approval processes, thus boosting developer productivity.

Some useful governance actions that you can enforce with Azure Policy include:

- Ensure that your team deploys Azure resources only to allowed regions.
- Enforce geo-replication rules to comply with your data residency requirements.
- Allow only certain virtual machine sizes for your cloud environment.
- Enforce the consistent application of taxonomic tags across resources.
- Recommend system updates on your servers.
- Allow multifactor authentication for all subscription accounts.
- Require resources to send diagnostic logs to an Azure Monitor Logs workspace.

Azure Policy evaluates your resources and highlights resources that aren't compliant with the policies that you create. Azure Policy can also prevent noncompliant resources from being created. Azure Policy comes with built-in policy and initiative definitions for Storage, Networking, Compute, Security Center, and Monitoring. For example, if you define a policy that only allows a certain size for the virtual machines (VMs) to be used in your environment, that policy is invoked when you create a new VM and whenever you resize existing VMs. Azure Policy also evaluates and monitors all current VMs in your environment, including VMs that were created before the policy was created.

In some cases, Azure Policy can automatically remediate noncompliant resources and configurations to ensure the integrity of the state of the resources. For example, if all resources in a specific resource group need to have the *AppName* tag with a value of *SpecialOrders*, Azure Policy can automatically apply that tag if it's missing. However, you maintain full control over your environment. If there's a particular resource that you don't want Azure Policy to automatically update, you can mark that resource as an exception, and the policy won't automatically update it.

Azure Policy also integrates with Azure DevOps by applying any continuous integration and delivery pipeline policies that pertain to the predeployment and post-deployment phases of your applications.

The objective when designing an Azure Policy should be to strike a balance between control and stability in one area with speed and results in the other. The main reason to maintain balance is to ensure that the environment remains highly manageable so that you can implement necessary levels of control to ensure governance without hindering operational efficiency or productivity. In establishing and enforcing these controls, you must take care that your speed to achieve efficiency isn't affected. Hence, achieving this balance between control and stability with speed and results, often requires thoughtful compromise, and it's necessary to carefully evaluate the potential impact before introducing new policies.

For more information, see [Microsoft Cloud Adoption Framework for Azure - Cloud Adoption Framework | Microsoft Learn](#).

5. Check your knowledge

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/check>

Check your knowledge

Completed

6. Introduction

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/overview>

Introduction

Completed

Azure Policy is a service that allows you to create, assign, and manage governance policies that enforce rules and effects over Azure resources to ensure that they stay compliant with your IT governance standards. These policies enforce various rules and effects on resources, ensuring that they adhere to corporate standards and service-level agreements. These policies are described in JSON format and are known as policy definitions. Azure Policy is crucial for enforcing organizational standards and assessing compliance on a large scale.

Azure Policy initiatives are a collection of Azure Policy definitions that are grouped together toward a specific goal or purpose. By consolidating multiple Azure policies into a single item, Azure Policy initiatives allow centralized control and enforcement of configurations across Azure resources.

Industry-specific organizations in sectors like government, public sector, finance, and others accelerate digital transformation and achieve better business outcomes. They can achieve this objective by adopting specific, sovereignty-focused Azure Policy initiatives to address the complexity of compliance with national and regional regulatory requirements.

Customers can create Azure Policy initiatives that aid in customizing deployments to reduce the time needed to audit environments and help meet established regulatory compliance frameworks and government requirements. The initiatives help public sector partners and customers create cloud guardrails and enforce specific regulations effectively. They can layer policy initiatives to help form a complete solution for their specific needs, and they can use deployment automation to ensure consistency, best practices, and save time.

In this module, you learn how Azure Policy can help do the common tasks related to creating, assigning, and managing policies across your organization, such as:

- Assign a policy to enforce a condition for resources you create in the future
- Create and assign an initiative definition to track compliance for multiple resources
- Resolve a noncompliant or denied resource
- Implement a new policy across an organization

Important

Organizations are wholly responsible for ensuring their own compliance with all applicable laws and regulations. The information provided in this document does not constitute legal advice, and organizations should consult their legal advisors for any questions regarding regulatory compliance.

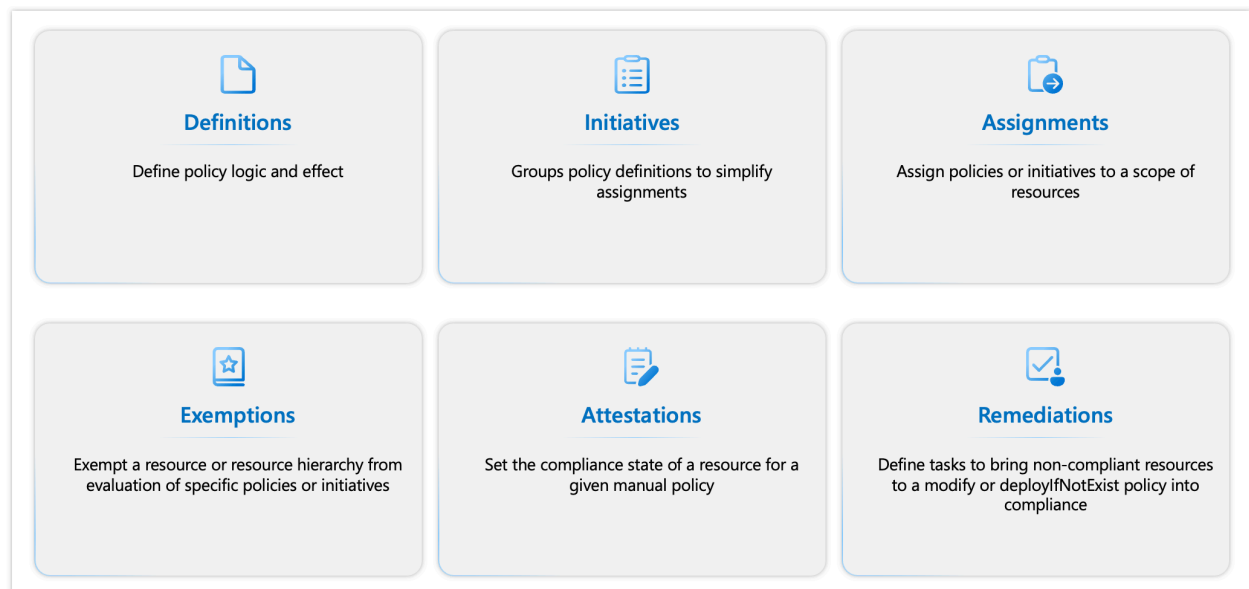
7. Azure Policy resources

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/azure-policy-resources>

Azure Policy resources

Completed

Azure Policy enforces organizational standards and assesses compliance at scale. It evaluates Azure resources and actions by comparing their properties to business rules, providing an aggregated view of the environment's overall state. This policy allows for detailed analysis down to each resource and policy level with granularity. Six policy resources are available in Azure, and multiple different concepts apply to these Azure policy resources.



Definitions

Azure Policy definitions describe resource compliance conditions and the effect to take if a condition is met. Several settings determine which resources are evaluated by any Azure Policy. You explore these settings in the next unit, **Azure Policy definitions**. The primary concept to which these settings can be applied is scope.

Scope in Azure Policy is the same as the levels of hierarchy for governance in Azure. Four levels of management scope are under the root tenant: the management groups, subscriptions, resource groups, and resources. The definition might be saved in a management group or a subscription. The definition location determines the scope to which the initiative or policy can be assigned. An assignment also has several properties that set a scope. The use of these properties determines which resource for Azure Policy to evaluate and which resources count toward compliance

You can apply management settings at any of these levels of scope. The level that you select determines how widely the setting is applied. Lower levels inherit settings from higher levels. For more information, see [Scope in Azure Policy](#).

Initiatives

Azure Policy initiatives, also known as a policy set, allow you to group several policy definitions to simplify assignments and management because you work with the initiatives as a single item. Initiatives offer a streamlined and automated approach to governance, allowing organizations to manage and monitor compliance at scale. The initiative definition contains all policy definitions to help track your compliance state for a larger goal, such as organizational compliance goals or compliance with regulatory frameworks. For example, multiple tagging policy definitions can be grouped into a single initiative, and rather than assigning each policy individually, you can apply the initiative to Azure resources. A JSON can be used to create a policy initiative definition. For more information, see [Azure Policy initiative definition structure](#).

For more information about Azure Policy built-ins and patterns, see [Azure Policy samples](#).

Built-in policy is a type of policy definition that's generated by Azure Resource Providers and is available by default. A group of such policy definitions is known as a **built-in initiative**. Azure built-in policy initiatives are a powerful tool set that enables centralized control across Azure resources and enforcement of specific configurations. These initiatives comprise a collection of policy definitions and support compliance with various regulatory frameworks, industry standards, and security best practices.

For a list of built-in policies and initiatives, see [Policies](#) and [Initiatives](#).

Custom policy is a type of policy definition that's written by a policy user when no built-in policy maps to your requirements. A group of such policy definitions is known as a **custom initiative**. Azure Policy custom initiatives help you to tailor a set of policies specifically to your organization's unique requirements, giving you control to enforce the standards and rules that best fit your environment. Microsoft for Sovereignty makes several custom policy initiatives and compliance mappings accessible through the [industry-policy-portfolio](#) repository on GitHub.

Microsoft for Sovereignty initiatives and compliance mappings, which expand on the Azure built-in initiatives, help you automate policy enforcement and foster a robust governance framework that reduces the risk of noncompliance. Further, the initiatives also strengthen data protection measures. Organizations can use the large suite of available regulatory compliance built-in initiatives while we continue to expand on other frameworks. These initiatives are available as Azure built-in and custom policy initiatives.

For more information, see [Microsoft for Sovereignty policy portfolio](#).

Assignments

Policy assignments define which resources are evaluated by a policy definition or initiative. Policy assignments can be done in the portal, an API call, or through the command line interface.

Policy and initiative definitions are deployed to a definition location (management group or subscription). This location determines the scope to which the initiative or policy can be assigned. The location should be the resource container shared by all resources that you want to use the policy definition on. For more information, see [Definition location](#).

Policies and initiatives are assigned to a specific scope (management group, subscription, or resource group). While doing so, you can define several optional aspects, including the resource scope and policy definition.

- Optional *resource selectors* to allow gradual rollout based on resource location or type.
- Optional *overrides* to change the effect of a policy definition without modifying the underlying definition.
- *enforcementMode* can be disabled to support "what-if" scenarios without changing the definition, which is equivalent to changing the definition to an audit effect mode, but a way to

do it at assignment level. For example, if the policy has *Deny* effects, that denial isn't effective, but you can still view the result of the compliance evaluation of that policy.

- Optional *excluded scopes* to exclude inner containers or resources from the assignment scope.
- *Noncompliance messages* can be defined.
- *Parameters* can be assigned values.
- If you have a policy with the *deployIfNotExists* effect type, a *managed identity* can be assigned (system-assigned or user-assigned) to turn on remediation actions. An assignment has several properties that set a scope. The use of these properties determines which resource for Azure Policy to evaluate and which resources count toward compliance. These properties map to the following concepts:
 - **Inclusion** - For more information, see [Azure Policy assignment structure](#).
 - **Exclusion** - For more information, see [Azure Policy assignment structure excluded scopes](#).

Exemptions

Use the Policy exemptions feature to *exempt* a resource hierarchy or an individual resource from evaluation of initiatives or definitions. Resources that are *exempt* count toward overall compliance but can't be evaluated or have a temporary waiver. They're created as a child object on the resource hierarchy, or the individual resource granted the exemption.

Policy exemptions aren't created during assignment time, but after, and the effect is still the same as an excluded scope. Two exemption categories exist and are used to group exemptions:

- **Mitigated** - The exemption is granted because the policy intent is met through another method.
- **Waiver** - The exemption is granted because the noncompliance state of the resource is temporarily accepted.

For more information about policy exemption, see [Azure Policy exemption structure](#).

Attestations

Policy attestations are used by Azure Policy to set compliance states of resources or scopes targeted by [manual policies](#). Each applicable resource requires one attestation for each manual policy assignment. For ease of management, manual policies should be designed to target the scope that defines the boundary of resources whose compliance state needs to be attested.

For more information, see [Azure Policy attestation structure](#).

Remediations

The policy remediation task feature is used to bring resources into compliance based on a definition and assignment. Resources that are noncompliant to a *modify* or *deployIfNotExists* definition assignment can be brought into compliance by using a remediation task. Resources that are newly

created or updated that are applicable to a *deployIfNotExists* or *modify* definition assignment are automatically remediated.

For more information, see [Azure Policy remediation task structure](#).

8. Azure Policy definitions

<https://learn.microsoft.com/en-us/training/modules/sovereignty-policy-initiatives/azure-policy-definitions>

Azure Policy definitions

Completed

Azure Policy definition describes resource compliance conditions and the action or effects that take place if those conditions are met. The policy consists of two parts:

- A **condition** that compares a resource property field or a value, accessed by using aliases, to a required value.
- The **effect** determines what happens when the policy rule is evaluated to match the condition. For each new resource, an updated resource, or an existing resource, the effects behave differently.

Anatomy of a policy definition

You use JSON to create a policy definition that contains the elements shown in the following table.

Element	Description	Properties or values
<i>displayName</i> (string, max 128 characters)	Used to identify the policy definition.	
<i>description</i> (string, max 512 characters)	Provides context for when the definition is used.	
<i>policyType</i> (read-only string)	Indicates the origin of the policy definition. This property can't be set, but SDK returns three values which are visible in the portal.	• Built in: Provided and maintained by Microsoft. • Custom: Custom definitions created by the customer. • Static: Regulatory Compliance policy with Microsoft ownership.

Element	Description	Properties or values
<i>mode (string)</i>	Configured depending on the target of the policy: an Azure Resource Manager property or a Resource Provider property.	- Resource Manager Modes: - On All: Evaluates resource groups, subscriptions, and all resource types. - Indexed: Evaluates resource groups, subscriptions, and all resource types. - Resource Provider Modes (limited to Built in policies and fully supported): - Microsoft.Kubernetes.Data - Microsoft.KeyVault.Data - Microsoft.Network.Data - Resource Provider Modes (limited to Built in policies and in preview mode): - Microsoft.ManagedHSM.Data - Microsoft.DataFactory.Data
<i>version (string, optional)</i>	Built-in policy definitions can host multiple versions with the same definitionID. If no version number is specified, all experiences show the latest version of the definition.	
<i>metadata (object, optional, max 1,024 characters)</i>	Stores information about the policy definition.	Common properties (for Built-in policies): <i>version (string)</i>: Tracks details about the version of the contents of a policy definition. <i>category (string)</i>: Determines under which category in the Azure portal the policy definition is displayed. <i>preview (Boolean)</i>: True or false flag that indicates if the policy definition is in preview. <i>deprecated (Boolean)</i>: True or false flag that indicates if the policy definition is deprecated. <i>portalReview (string)</i>: Determines if parameters require review in the portal.
<i>parameters (object, optional)</i>	Help simplify your policy management by reducing the number of policy definitions. By including parameters in a policy definition, you can reuse that policy for different scenarios by using different values.	Properties: - name - type (String, Array, Object, Boolean, Integer, Float, DateTime) - metadata (description, displayName, strongType, assignPermissions) - defaultValue - allowedValues - schema

Element	Description	Properties or values
<i>policyRule</i> (object)	The effect of a policy is defined in the <i>policyRule</i>. The policy rule consists of the <i>if</i> and <i>then</i> blocks. • In the <i>if</i> block, you define one or more conditions that specify when the policy applies. • In the <i>then</i> block, you define the effect that happens when the <i>if</i> conditions result true.	

```
{
  "displayName": "Allowed locations",
  "description": "This policy enables you to restrict the locations your organization can use to deploy resources.",
  "policyType": "BuiltIn",
  "mode": "Indexed",
  "metadata": {
    "version": "1.0.0",
    "category": "General"
  },
  "parameters": {
    "listOfAllowedLocations": {
      "type": "Array",
      "metadata": {
        "description": "The list of locations that can be specified when deploying resources.",
        "strongType": "location",
        "displayName": "Allowed locations"
      }
    }
  },
  "policyRule": {
    "if": {
      "allOf": [
        {
          "field": "location",
          "notIn": "[parameters('listOfAllowedLocations')]"
        },
        {
          "field": "location",
          "notEquals": "global"
        },
        {
          "field": "type",
          "notEquals": "Microsoft.AzureActiveDirectory/b2cDirectories"
        }
      ]
    },
    "then": {
      "effect": "deny"
    }
  }
}
```



```
}
```

In the given example, *b2cDirectories* is excluded from the policy logic because its location field isn't a region (it can be "United States," "Europe," "Asia Pacific," or "Australia"). This logic can be enforced with a separate policy.

Logical operators and conditions (*if* blocks)

The first part of the *policyRule* in an Azure Policy definition is the *if* block. This block defines the conditions for which the policy evaluates the resources. A policy definition can contain several conditional statements. Depending on your evaluation requirements, you might or might not need each statement to be true, and you might only need some of them to be true.

Logical operators supported in the *if* block

In the *if* condition, you can put different logical operators.

Operator	Type	Description
<i>not</i>	{condition or operator}	The <i>not</i> syntax inverts the result of the condition.
<i>allOf</i>	[[{condition or operator}, {condition or operator}]	The <i>allOf</i> syntax (like the logical <i>and</i> operation) requires all conditions to be true.
<i>anyOf</i>	[[{condition or operator}, {condition or operator}]	The <i>anyOf</i> syntax (like the logical <i>or</i> operation) requires one or more conditions to be true.

The following example shows use of the *allOf* logical operator:

```
{
  "if": {
    "allOf": [
      {
        "field": "location",
        "notIn": "[parameters('listOfAllowedLocations')]"
      },
      {
        "field": "location",
        "notEquals": "global"
      },
      {
        "field": "type",
        "notEquals": "Microsoft.AzureActiveDirectory/b2cDirectories"
      }
    ]
  },
  "then": {
    "effect": "deny"
  }
}
```

```
}  
}
```

Nested logical operations

Logical operations are optional and can be nested to create complex scenarios.

The following example shows a *not* operation nested in an *allOf* operation:

```
"if": {  
  "allOf": [  
    {  
      "not": {  
        "field": "tags",  
        "containsKey": "application"  
      }  
    },  
    {  
      "field": "type",  
      "equals": "Microsoft.Storage/storageAccounts"  
    }  
  ]  
},
```

Conditions

Properties like fields, values, or counts can be evaluated within a condition.

Fields	Conditions that evaluate whether the values of properties in the resource request payload meet certain criteria can be formed by using a field expression.	Name, fullName, kind, type, location, ID, identity.type, tags, tags['tagName'], property aliases
Value	Conditions that evaluate whether a value meets certain criteria can be formed by using a value expression.	
Count	Conditions that count how many members of an array meet certain criteria can be formed by using a count expression.	- Field count, value count - The current () function returns the value of the array member that's being evaluated

The condition in an Azure Policy assesses whether the evaluated values for the properties, such as Fields, Value, or Count, meets certain criteria. If the result of a function is an error, the policy results in a deny effect. This result can be avoided while testing by disabling *enforcementMode* in the assignment. For more information, see [Enforcement Mode](#).

Evaluation criteria	Value type
<i>equals</i>	stringValue
<i>notEquals</i>	stringValue

Evaluation criteria	Value type
<i>like</i>	stringValue
<i>notLike</i>	stringValue
<i>match</i>	stringValue
<i>notMatch</i>	stringValue
<i>matchInsensitively</i>	stringValue
<i>notMatchInsensitively</i>	stringValue
<i>contains</i>	stringValue
<i>notContains</i>	stringValue
<i>In</i>	["stringValue1", "stringValue2"]
<i>notIn</i>	["stringValue1", "stringValue2"]
<i>containsKey</i>	keyName
<i>notContainsKey</i>	keyName
<i>less</i>	dateValue
<i>less</i>	stringValue
<i>less</i>	intValue
<i>lessOrEquals</i>	dateValue
<i>lessOrEquals</i>	stringValue
<i>lessOrEquals</i>	intValue
<i>greater</i>	dateValue
<i>greater</i>	stringValue
<i>greater</i>	intValue
<i>greaterOrEquals</i>	dateValue
<i>greaterOrEquals</i>	stringValue
<i>greaterOrEquals</i>	intValue
<i>exists</i>	bool

The following JSON is an example of using conditions:

```
{
  "if": {
    "allOf": [
      {
        "value": "[resourceGroup().name]",
        "like": "*netrq"
```

```

    },
    {
      "field": "type",
      "notLike": "Network/*"
    }
  ]
},
"then": {
  "effect": "deny",
  "details": {
    "count": {
      "field": "Microsoft.Network/virtualNetworks/addressSpace.addressPrefixes[*]",
      "where": {
        "value": "[ipRangeContains('10.0.0.0/24', current('Microsoft.Network/virt
        "equals": "greater"
      }
    }
  }
}
}
}
}

```

Policy functions

Functions can be used to introduce extra logic into a policy rule. They're resolved in the policy rule of a policy definition and in the parameter values that are assigned to the policy definitions in an initiative.

The Resource Manager template functions are available to use in a policy rule except a [few policy functions and user-defined functions](#).

The `utcNow()` function is available to use in a policy rule but differs from use in an Azure Resource Manager template (ARM template). Unlike an ARM template, this function can be used outside `defaultValue`. It returns a string set to the current date and time in Universal ISO 8601 DateTime format `yyyy-MM-ddTHH:mm:ss.fffffffZ`.

The following table describes the functions that are only available in policy rules.

Function	Description
<code>addDays(dateTime, numberOfDaysToAdd)</code>	<code>dateTime</code> : [Required] string - String in the Universal ISO 8601 DateTime format 'yyyy-MM-ddTHH:mm:ss.FFFFFFFZ'. <code>numberOfDaysToAdd</code> : [Required] integer - Number of days to add.
<code>Field(fieldName)</code>	<code>fieldName</code> : [Required] string - Name of the field to retrieve. - Returns the value of that field from the resource evaluated by the <i>If</i> condition. <code>field</code> is primarily used with <code>auditIfNotExists</code> and <code>deployIfNotExists</code> to reference fields on the resource that are being evaluated.

Function	Description
<code>requestContext().apiVersion</code>	Returns the API version of the request that triggered policy evaluation. This value is the API version that was used in the PUT/PATCH request for evaluations on resource creation/update. The latest API version is always used during compliance evaluation on existing resources.
<code>policy()</code>	Returns the following information about the policy that's being evaluated. Properties can be accessed from the returned object. <pre>"assignmentId": "", "definitionId": "", "setDefinitionId": "", "definitionReferenceId": ""</pre>
<code>ipRangeContains(range, targetRange)</code>	<code>range</code> : [Required] string - String specifying a range of IP addresses to check if the <i>targetRange</i> is within range. <code>targetRange</code> : [Required] string - String specifying a range of IP addresses to validate as included within the <i>range</i>. Returns a <i>boolean</i> for whether the <i>range</i> IP address range contains the <i>targetRange</i> IP address range. Empty ranges or mixing between IP families isn't allowed and results in evaluation failure.
<code>current(indexName)</code>	Special function that can only be used inside count expressions.

Effect types (*then* blocks)

The second part of the *policyRule* in an Azure Policy definition is the *then* block. This block defines the effect that takes place when the policy rule is evaluated to match the condition resources. More than one effect can be valid for a given policy definition. Parameters are often used to specify allowed effect values (*allowedValues*) in such cases so that a single definition can be more versatile during assignment. Resource properties and logic in the policy rule can determine whether a certain effect is considered valid to the policy definition.

Effect	Description	Type
<i>disabled</i>	The <i>disabled</i> effect is a way to deactivate the policy. If a policy definition has <i>Disabled</i> as its effect, any assignments of that policy aren't active. This effect is checked first to determine if the policy rule should be evaluated. This flexibility makes it possible to deactivate a single assignment instead of deactivating all of that policy's assignments.	Synchronous evaluation
<i>append</i>	The <i>append</i> effect is used to add more fields to the requested resource during creation or update. It's mostly obsolete because <i>Modify</i> can also be used to add fields to the request.	Synchronous evaluation
<i>modify</i>	The <i>modify</i> effect is used to add, update, or remove properties or tags on a subscription or resource during creation or update. It allows	Synchronous evaluation

Effect	Description	Type
	Azure Policy to modify requests to Azure Resource Manager by altering fields to ensure compliance.	
<i>deny</i>	The <i>deny</i> effect is used to prevent a resource request that doesn't match defined standards through a policy definition and fails the request.	Synchronous evaluation
<i>denyAction</i>	The <i>denyAction</i> effect is used to block requests based on intended action to resources at scale. Currently, the only supported action is DELETE.	Synchronous evaluation
<i>audit</i>	The <i>audit</i> effect is used to create a warning event in the activity log when you're evaluating a noncompliant resource, but it doesn't stop the request.	Asynchronous evaluation
<i>auditIfNotExists</i>	The <i>auditIfNotExists</i> effect allows the auditing of resources that are related to the resource that matches the <i>if</i> condition but doesn't have the properties specified in the details of the <i>then</i> condition.	Asynchronous evaluation
<i>deployIfNotExists</i>	The <i>deployIfNotExists</i> policy definition runs a template deployment when the condition is met. It can trigger deployment of a related resource based on the compliance state of the currently evaluated resource.	Asynchronous evaluation
<i>manual</i>	The <i>manual</i> effect allows you to self-attest the compliance of resources or scopes. When a policy definition with <i>Manual</i> effect is assigned, you can set the compliance states of targeted resources or scopes through custom attestations.	Manual attestation

The following list provides general guidance around interchangeable effects:

- *audit*, *deny*, and either *modify* or *append* are often interchangeable.
- *auditIfNotExists* and *deployIfNotExists* are often interchangeable.
- *manual* isn't interchangeable.
- *disabled* is interchangeable with any effect.

Multiple policies can be assigned to a single resource at the same scope or at different scopes. Each policy mostly has a different effect defined. The condition and effect for each policy is independently evaluated. The net result of layering policy definitions is considered **cumulative most restrictive**.