# Azure Policy Governance

# Introduction

Policy-driven governance is one of the Azure Landing Zones design principles of the Cloud Adoption Framework. By leveraging Azure Policy to provide security- and service management guardrails for Azure Landing Zones, more autonomy for application teams can be provided. This document contains an approach to a naming convention and governance model for Azure Policy.

# Table of Contents

* [Introduction](#introduction)
* [Envisioning](#envisioning)
  + [Principles](#principles)
  + [Requirements](#requirements)
  + [Environment](#environment)
  + [Lifecycle management](#lifecycle-management)
* [Repository](#repository)
  + [Folder Structure](#folder-structure)
  + [Filenames](#filenames)
* [Policy Definitions](#policy-definitions)
  + [Standard Policy Definitions](#standard-policy-definitions)
    - [Example standard Policy Definition](#example-standard-policy-definition)
    - [Commonly used 'Property of Azure Service' keywords](#Xed9963c1e1d5b8beee9da2d18bd2fb6c639612a)
* [Policy Initiatives](#policy-initiatives)
  + [Example](#example)
  + [Group Definitions in Initiatives](#group-definitions-in-initiatives)
* [Policy Assignments](#policy-assignments)
  + [Example](#example-1)
* [Versioning](#versioning)
  + [Changes to Policy definitions](#changes-to-policy-definitions)
  + [Changes to Policy Initiative definitions](#changes-to-policy-initiative-definitions)
* [Release notes](#release-notes)
  + [Policy definitions](#policy-definitions)
  + [Policy Initiatives](#policy-initiatives-1)
* [Development process Azure Policy artifacts](#Xcc655e4b577fe3612bef4488715d48d88ce2db3)
* [Testing Azure Policy artifacts](#testing-azure-policy-artifacts)
  + [Engineering environment](#engineering-environment)
  + [End-to-end test plan](#end-to-end-test-plan)

# Envisioning

When thinking about a naming convention, start with the end in mind, and by that I mean think about the principles that underlie your policy governance solution, what your requirements are, how it should fit your environment and how you want to implement lifecycle management.

In the sections below, I’ve listed the principles, requirements and scenarios that form the basis for my naming convention.

## Principles

* Policy Definitions are grouped per Azure Service in a Policy Initiative. Only Policy Initiatives are assigned.
* Major version updates require replacement of an existing Policy/Policy Initiative (see versioning)
* Minor and Patch updates are done ‘in place’
* Policy Definitions are always deployed at root management group scope. This allows the policies to be used in all lower-level management groups.
* Policy Initiatives/Definitions can be Assigned at multiple Management Group scopes with different parameters
* Implement policy-as-code
* The repository is the single-source-of-truth

## Requirements

* Phased rollout of Policy assignments
  + I want to be able to use a phased approach to the rollout of Policy assignments that can impact workloads (e.g. dev, test, acceptance, production)
* Archiving of Policy artifact versions
  + Retain n-1 major version in repo for archiving and rollback
* Rollback of a Policy
  + Retain n-1 major version in repo for archiving and rollback
* Major versions have a release note associated with a Policy Definition and Initiative
  + Especially in a complex environment, use release notes to keep track of changes
* Compliance state for a specific service or topic must be easy to monitor in the Azure Compliance Dashboard

## Environment

* Single Azure LandingZone hierarchy
* Single Azure LandingZone hierarchy with multiple business units
* Global Azure LandingZone hierarchy

## Lifecycle management

There’s more to life cycle management, than just these three topics, but within the scope of naming convention these three topics are most relevant.

* Adding new Policy (Initiative) Definition
* Modify a Policy (Initiative) Definition
* Remove Policy (Initiative) Definition

[Back to top](#introduction)

# Repository

The structure of the repository is dependent on the selected Policy deployment engine, but this chapter contains an example approach to the structure of a repository.

## Folder Structure

In the root are the folders ‘Assignments’ and ‘Definitions’. To support delegation of control, these could also be separate repositories.

Policy Definitions are in the ‘PolicyDefinitions’ folder. Each Azure Service/topic has a separate sub-folder to keep things manageable. The Policy Definitions are grouped into Policy Initiatives in the ‘InitiativeDefinitions’ folder. Again, each subfolder represents a Policy Initiative.

The figure below shows a sample structure when using the PolicyAsCode framework from the IacS project:

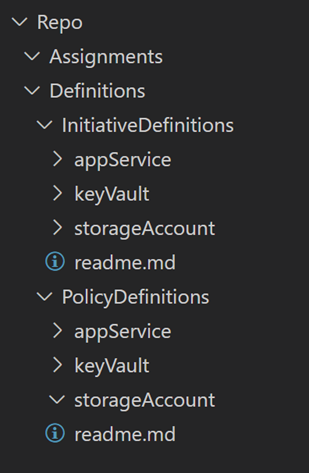


Figure 1 - example folder structure

This figure shows a sample folder structure when using the [EnterprisePolicy as Codesolution](https://github.com/Azure/enterprise-azure-policy-as-code):

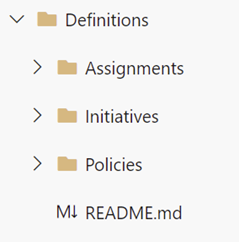


Figure 2 - example EPAC folder structure

## Filenames

As the policy artifacts are stored as code, we need a naming convention for the files in the repository.

| **Artifact** | **Standard** | **Example(s)** |
| --- | --- | --- |
| PolicyDefinition | <prefix>-<service/scope>-<effect>-<description>-v<major version>.json | cto-sa-audit-firewall-settings-v1.json cto-sa-dine-diagnostic-settings.v1.json |
| PolicyInitiativeDefinitions | <initiative>-<scope/service>-\v.json | initiative-keyvault-v1.json initiative-platform-locations.v1.json |

The table below contains the rationale for the different components of the naming convention:

| **Naming convention** | **Rationale** |
| --- | --- |
| Policy Definition file name is equal to the ‘name’ property of the Policy Definition. | This creates a 1:1 relation with the Policy Definition and helps to locate files in the repository. |
| Prefix (can follow any taxonomy) | A prefix separates built-in from custom policy artifacts. It allows for easy filtering/location in the Azure Portal. |
| Policy Initiative Definition file names start with the prefix ‘initiative’. | Optional. This separates initiative files from Policy Definition files to prevent confusion. |
| Service/scope[1] | Helps to determine which Azure Service or scope is affected by the Policy (Initiative). |
| Description | Short and concise description of the purpose of the policy. |
| Major version | Only major changes require a new policy definition (file). Minor and patch updates are applied within the current version. |

The table below contains the rationale for the different components of the naming convention:

**Note**

Any major update requires replacement of the affected Policy Definitions and/or Policy Initiative Definition. Therefore, the filename has a suffix with the major version to distinguish between other major versions of the same Policy artifact.

[Back to top](#introduction)

# Policy Definitions

A Policy Definition consists of multiple properties such as a 'name' and 'displayName'. Since we want everyone to understand the purpose of the Policy Definition, the following standard was defined for all relevant properties:

## Standard Policy Definitions

| Property | Standard | Example(s) |
| --- | --- | --- |
| name (max. 64 char) | <prefix>-<service/scope>-<effect>-<description>-v<major version> | cto-sa-audit-firewall-settings-v1 cto-sa-dine-diagnostic-settings-v1 |
| displayName (max. 128 char) | <prefix> - <service/scope> - <effect> <description> v<major version> | CTO - Storage Account - Audit Firewall Settings v1 CTO - Network - Dine Network Watcher v1 |
| description (max 512 char) | Concise explanation of what the Policy Definition evaluates | This Azure Policy evaluates the Firewall configuration of a Key Vault |
| version (metadata) | Use ‘semantic versioning’ to determine the version number | 1.0.0 |
| category (metadata) | Scope / Azure Service | Key Vault Platform |

The table below contains the rationale for the different components of the naming convention:

| Naming convention | Rationale |
| --- | --- |
| Name | The name property of a Policy Definition must be unique and is used to reference a specific PolicyDefinitionId To support multiple major versions of a Policy, the major version is part of the name. A prefix helps to distinguish between built-in and your custom policies and allows for easy filtering in the Compliance dashboard. Try to limit the prefix to 3-4 char max. |
| displayName | The displayName is a ‘human friendly’ version of the name property as it’s the displayName that is visible in the Azure Portal. |
| Descirption | Required to understand the purpose/effect of the Policy definition. |
| Major version | Only major changes require a new policy definition. Minor and patch updates are applied within the current version. |
| Version (metadata) | Sematic versioning is used to track minor and patch updates within the major version. Updating the version in the Policy definition helps when developing or troubleshooting. The Policy Definition version is tied to the content in ‘release notes’ (optional). |
| Catagory (metadata) | Can be used to designate the Azure Service the policy affects (also part of the name). Can be used to designate scope (e.g. regional/global). Can be used to relate to either Security Control or Service Management Control. |

### Example standard Policy Definition

When following the standards described above, a standard Policy Definition should look like the example shown below:

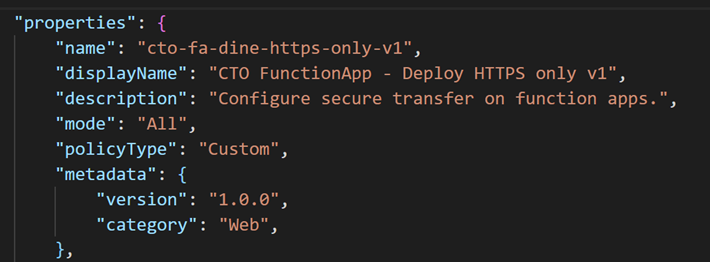


Figure 3 - Example of Policy Definition

### Commonly used 'Property of Azure Service' keywords

Some keywords are used across multiple Azure Services. To maintain consistency, the table below lists a number of common keywords:

| Property | Standard | Example |
| --- | --- | --- |
| Diagnostic Settings | diagnostic-settings | cto-kv-dine-diagnostic-settings-v1 |
| Networking | firewall-settings | cto-acr-audit-firewall-settings-v1 |
| SSL/TLS Settings | tls-settings | cto-appsvc-append-tls-settings-v1 |
| Azure Trusted Services | trusted-services | cto-sa-deny-trusted-services-v1 |

[Back to top](#introduction)

# Policy Initiatives

A Policy Initiative groups a set of Policy Definitions related to a specific scope or Azure Service. The table below contains the standards defined for Policy Initiatives:

| Property | Standard | Example(s) |
| --- | --- | --- |
| Name (max. 64 char) | <prefix>-<scope/service>-v<major version> | cto-keyvault-v1 cto-platform-v1 |
| displayName (max. 128 char) | <prefix> - <scope/service> v<major version> | CTO - Key Vault v1 CTO – Platform v1 |
| Description (max. 256 char) | States the scope the Policy Initiative applies to (for example Azure Service or platform service) | This Policy Set bundles all Policy Definitions for the Azure Key Vault Service |
| version (metadata) | Use Semantic Versioning to determine the version number | 1.0.0 |
| category (metadata) | Scope / Azure Service Key Vault | Platform |

The table below contains the rationale for the different properties of aPolicy Initiative Definition:

| Naming Convention | Rationale |
| --- | --- |
| name | The name property of a Policy Initiative Definition must be unique. To support multiple major versions of a Policy Initiative, the major version is part of the name. |
| displayName | The displayName is a ‘human friendly’ version of the name property as it’s the displayName that is visible in the Azure Portal. |
| Description | Required to understand the purpose/effect of the Policy definition. |
| Major version | Only major changes require a new policy definition. Minor and patch updates are applied within the current version. |
| Version (metadata) | Sematic versioning is used to track minor and patch updates within the major version. Updating the version in the Policy definition helps when developing or troubleshooting. The Policy Definition version is tied to the content in ‘release notes’ (optional). |
| Category (metadata) | Can be used to designate the Azure Service the policy affects (also part of the name). Can be used to designate scope (e.g. regional/global). Can be used to relate to either Security Control or Service Management Control. |

### Example

When following the standards described above, the Policy Initiative Definition should look like the example shown below:

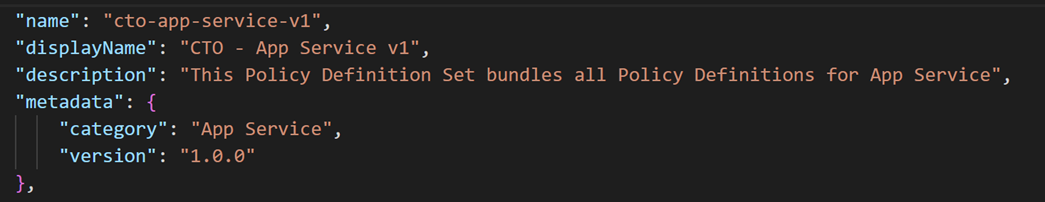


Figure 4 - Example Policy Initiative definition

### Group Definitions in Initiatives

Policy Definitions can be grouped together in a Policy Initiative across multiple axis. For example, you can create Policy Initiatives based on the Azure Service the policies apply to, or you can create Policy Initiatives for policies that address specific security controls. It’s this taxonomy that you’ll see in the Policy Compliance dashboard, so choose the structure that is most relevant to you.

In my example, I structured my initiatives based on Azure Service or platform topic (e.g. networking).

1. **Platform Initiatives**, containing:
   * Policy Definitions that cannot easily be grouped under Resource Type-specific Initiatives,
     + For example, an initiative to enforce specific Azure regions
   * Policy Definitions enforcing Platform-wide, Platform Security Standard
     + For example, an initiative to enforce Network Security Controls or Security Center Logging
   * Policy Definitions enforcing similar standards across *all* different Azure Resource Types, requiring a separate Definition for each Resource Type
     + For example, an initiative to apply default diagnostic settings on all Azure Resource Types
     + These Definitions are grouped as Platform Initiatives to avoid a large number of Resource Type-specific Initiatives containing only a single Definition
2. **Resource Type-specific Initiatives**, contain all Policy Definitions that belong to a specific Azure Resource Type.
   * Resource Type-specific Initiatives are created for Resource Types that will be governed by a number Policy Definitions

Some examples of Platform Initiatives are the following:

* Platform Diagnostics
* Platform Locations
* Platform Networking
* Platform Defender4Cloud
* Platform Sentinel
* Platform Tags

When a new Policy Definition is created, it should be placed in an Initiative following the evaluation below:

1. If a suitable Platform Initiative is available, it is placed there
2. If no suitable Platform Initiative is available, but a fitting Resource-Type Initiative is available, it is placed there
3. If neither a suitable Platform Initiative, nor a fitting Resource Type-specific Initiative is available, a new initiative should be created
   * This can be either a Platform or a Resource Type-specific Initiative
   * The guidance above can be used to decide which type of Initiative is most suitable

[Back to top](#introduction)

# Policy Assignments

There's a difference between Policy (Initiative) Definition creation and assignment. After creation, a Policy (Initiative) Definition is available for assignment, but the policy is not yet active. Only after assigning a policy, possibly with required parameters, it becomes 'active' for the specified scope. When set up correctly, a Policy Initiative definition can be assigned multiple times at different scopes with different parameters.

### Example

In my previous examples, Policy (Initiative) definitions all use the prefix 'CTO', but since there can be multiple policy assignments at different scopes, a different prefix can be used for different scopes. Consider the example below: the diagnostic Policy Definitions are grouped in the ‘CTO – Foundation Diagnostics v1’ Policy Initiative definition. The Policy Initiative definition ‘CTO – Foundation Diagnostics v1’ is then assigned multiples times at different Management Group scopes, using a different prefix and different parameter values.

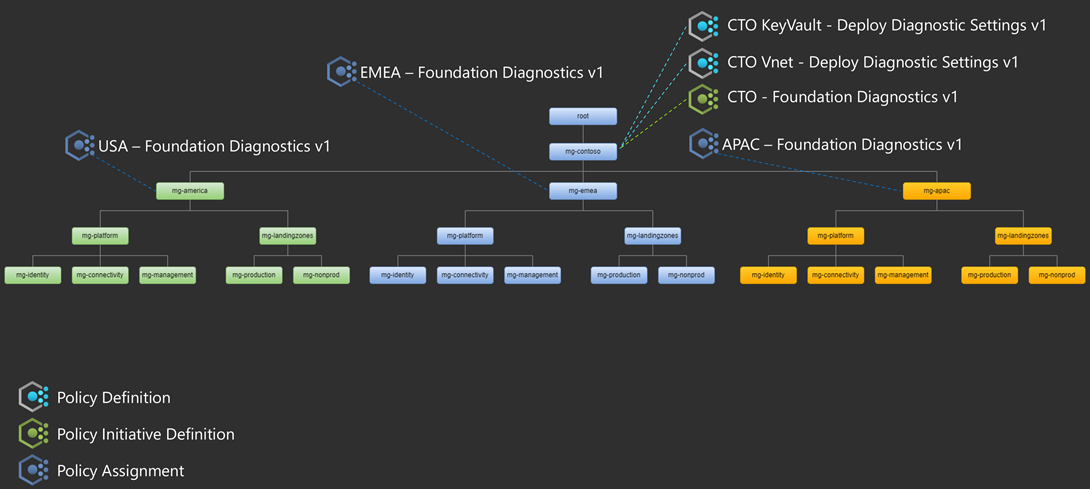
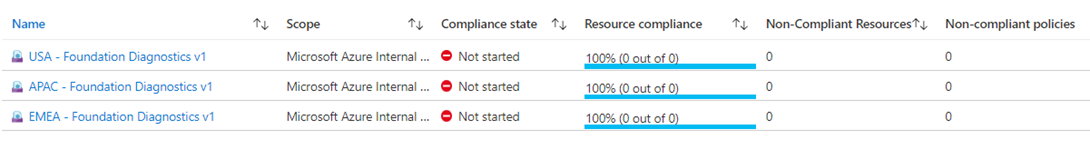


Figure 5 - Example policy assignments

In the Policy compliance dashboard, you will see something like this:



The naming convention of Policy Assignments is very similar to the Policy Initiative convention. Difference being the option to use a different prefix and the limit of 24 characters for the assignment ‘name’ property.

| Property | Standard | Example(s) |
| --- | --- | --- |
| Name (max. 24 char) | <prefix>-<scope/service>-v<major version> | cto-keyvault-v1 emea-platform-v1 |
| displayName (max. 128 char) | <prefix> - <scope/service> v<major version> | CTO - Key Vault v1 CTO – Platform v1 |
| Description (max. 256 char) | States the scope the Policy Initiative applies to (for example Azure Service or platform service) | This Policy Set bundles all Policy Definitions for the Azure Key Vault Service |

| Naming convention | Rationale |
| --- | --- |
| name | The name a Policy assignment must be unique. To support multiple major versions of a Policy Assignment, the major version is part of the name. The prefix for a Policy Assignment, can be different as the one used for your Policy (Initiative) Definitions to enable you to distinguish between different assignments of the same definition (see example). Take note of the limitation of 24 characters, which is much less than the usual 64 characters and also a good reason to limit you prefix to 3-4 chars. |
| displayName | The displayName is a ‘human friendly’ version of the name property as it’s the displayName that is visible in the Azure Portal. Description Required to understand the purpose/effect/scope of the Policy assignment. |

[Back to top](#introduction)

# Versioning

Versioning of Policy (Initiative) definitions[1] is based on Semantic Versioning. A Semantic Version contains a Major (**1**.0.0), Minor (1.**0**.0) and Patch (1.0.**0**) part, which are used to define the impact of a change. The table below describes which part of the version needs to be updated for different changes.

## Changes to Policy definitions

| Change | Part | Reasoning |
| --- | --- | --- |
| Rule logic changes | Major | Changes the result of policy evaluation |
| ifNotExists existence condition changes | Major | Changes the result of policy evaluation |
| Major changes to the effect of the policy (i.e. adding a new resource to a deployment) | Major | Either requires different structure of the policy (e.g. audit -> auditIfNotExists), or changes the result of the deployment (E.g. deployIfNotExists policy) |
| Changes to effect details that don't meet the major version criteria | Minor | Changes that have no impact on policy evaluation |
| Adding new parameter allowed values | Minor | Does not change the result of policy evaluation, but rather expands the scope (e.g. add additional locations to the allowedLocations parameter) |
| Adding new parameters (with default values) | Minor | Replacing a hard coded value with a parameter that has the hard coded value as default value |
| Other minor changes to existing parameters | Minor | Changes to metadata |
| String changes (displayName, description, etc…) | Patch | Does not affect the result of policy evaluations |
| Other metadata changes | Patch | Changes to the policy definition metadata |

## Changes to Policy Initiative definitions

| Change | Part | Reasoning |
| --- | --- | --- |
| Addition or removal of a policy definition from the policy set | Major | Doesn’t require unassignment of PolicySet. However, the added policy can impact the existing environment depending on the effect of the policy. |
| Adding new parameter allowed values | Minor | See changes to Policy |
| Adding new parameters (with default values) | Minor | See changes to Policy |
| Other minor changes to existing parameters | Minor | See changes to Policy |
| tring changes (displayName, description, etc…) Patch | See changes to Policy |  |
| Other metadata changes | Patch | See changes to Policy |

[1]: Source: <https://github.com/Azure/azure-policy/blob/master/built-in-policies/README.md#versioning>

[Back to top](#introduction)

# Release notes

In a complex environment, multiple stakeholders can be making changes to Policy (Initiative) definitions and assignments, it's challenging to keep up to date with the changes. What has changed and do other stakeholders need to move to a new major version of a Policy Initiative? That's where release notes come in to help keep track of the changes.

Whenever a **production** update is implemented for a Policy (Initiative) definition, make sure a **release note** file is **created**, or **updated**, that describes the changes. A new release note is created for each major version.

## Policy definitions

The sample folder structure below shows how release notes can be added to the structure:

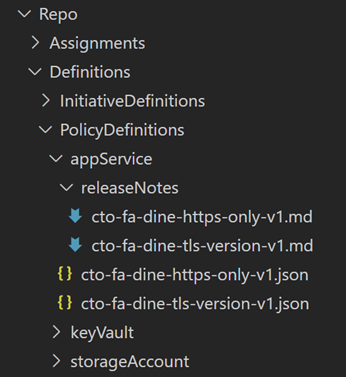


Figure 6 - Example folder structure for Policy definition release notes

The release note filename matches the filename of the Policy definition in a 1:1 relationship.

The screenshot below shows an example of a release note for a new major version of a Policy Definition. Any minor or bugfixes that are implemented afterwards are added to this release note:

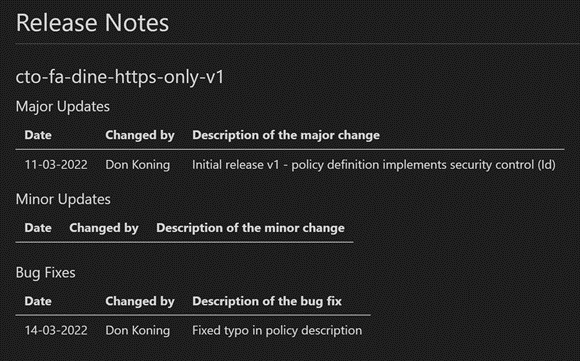


Figure 7 - Example Policy definition release note

## Policy Initiatives

A new release note is created for each major version and is placed in the subfolder of the Policy Initiative Category, as shown in the example below:

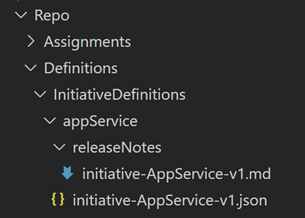
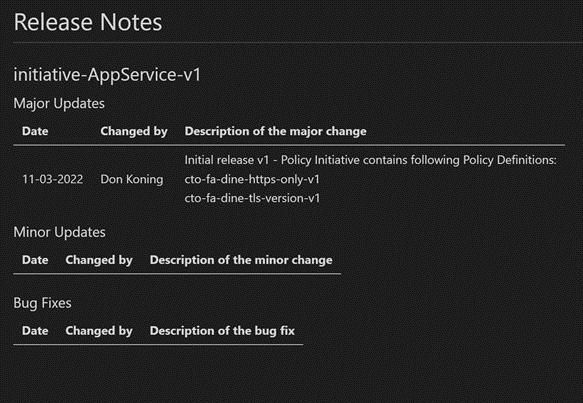


Figure 8 - Example folder structure for Policy Initiatives

The release note is named after the Policy Initiative definition file in a 1:1 relationship.

The screenshot below shows an example of a release note for a new major version of a Policy Initiative. Any minor or bugfixes that are implemented later are added to this release note:



[Back to top](#introduction)

# Development process Azure Policy artifacts

TBD (dependant on selected Azure Policy deployment solution)

[Back to top](#introduction)

# Testing Azure Policy artifacts

Azure Policy allows for platform-level desired state control, meaning a ‘Deny’ Policy on a Management Group scope can affect all deployments in the hierarchy below. Because of the implications a misconfiguration of Azure Policy may have, it is important to properly test the effect the Policy has on an environment and its products/services.

## Engineering environment

TBD

## End-to-end test plan

When developing Azure Policy objects, it is important to validate that they work as intended. As changes are merged to the ‘main’ branch after testing, it is recommended to document and include a reference to the test results in the Pull Request. As such, the reviewer will find it less difficult to review the Pull Request, and if everything checks out, approve it.

As part of a successful and complete test, consider taking the following steps:

1. Deploy two Azure Resources (one compliant and one incompliant) that you are trying to evaluate against in a Policy engineering environment
2. If existing ‘automation' exists, deploy the Azure Service using the automation to get a representative deployment of the service
3. Create a branch that you will use for the test use case. All changes/additions/modifications must be done from this branch so you do not impact other tests that might be on-going
4. Create the custom policy (Initiative) definitions
5. Assign the Policy Initiative scoped to the Resource Group where you deployed your two test resources
6. Trigger Policy evaluation
7. See if the evaluation is done correctly. Change settings to validate that the evaluation result changes as expected
8. Define test scenarios for your policy and document the test results
9. Proceed to fill the following checks:
   * If applicable: is the 'automated' deployment compliant by default?
   * Did you test all angles of the Policy?
   * Did you extract data about the test results?
10. Test complete
11. Clean-up test environment

[Back to top](#introduction)