Understanding Hierarchy Evaluation

When you set an organization policy (https://cloud.google.com/resource-manager/docs/organization-policy/overview) on a resource hierarchy node, all descendants of that resource hierarchy node inherit the organization policy by default. If you set an organization policy at the root organization node, then those restrictions are inherited by all child folders, projects, and resources.

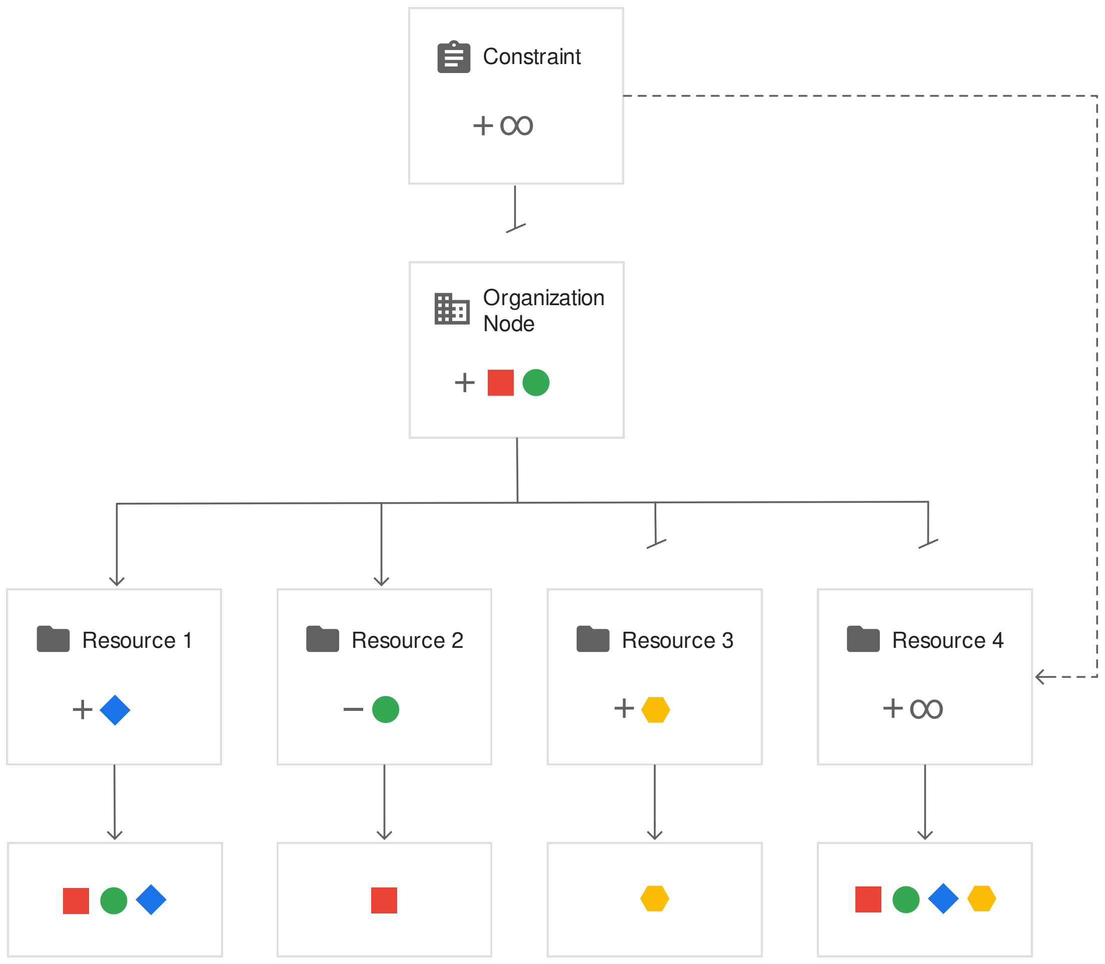
You can set custom organization policy on child nodes, which will overwrite or merge with the inherited policy based on the rules of hierarchy evaluation.

Before you begin

- Read the Understanding constraints (https://cloud.google.com/resource-manager/docs/organization-policy/understanding-constraints) page to learn about what a constraint is.
- Read the overview of the Organization Policy Service (https://cloud.google.com/resource-manager/docs/organization-policy/overview) to learn about how organization policy works.

Example hierarchy

In the resource hierarchy diagram below, each node sets a custom organization policy and defines whether it inherits its parent node's policy. The colored shapes represent the values that the organization policy allows or denies.



A **Constraint** is a definition of the behaviors that are controlled by an organization policy. The **Constraint** in the above example has a default behavior that allows all values. The nodes below it define custom policies that overwrite this behavior by allowing or denying values.

The effective policy on each node is evaluated based on the rules of inheritance. If a custom organization policy is not set, the node will inherit the default constraint behavior. If you do set an organization policy, your custom policy is used instead. In the above example, the **Organization Node** defines a policy that allows red square and green circle.

The resource nodes that are in the hierarchy below the **Organization Node** are evaluated as follows:

1. **Resource 1** defines a custom policy that sets `inheritFromParent` to `TRUE` and allows blue diamond. The policy from the **Organization Node** is inherited and merged with the custom

policy, and the effective policy evaluates to allow red square, green circle, and blue diamond.

2. **Resource 2** defines a custom policy that sets `inheritFromParent` to `TRUE` and denies green circle. Deny values always take precedence during policy reconciliation. The policy from the **Organization Node** is inherited and merged with the custom policy, and the effective policy evaluates to allow only red square.
3. **Resource 3** defines a custom policy that sets `inheritFromParent` to `FALSE` and allows yellow hexagon. The policy from the **Organization Node** is not inherited, so the effective policy evaluates to only allow yellow hexagon.
4. **Resource 4** defines a custom policy that sets `inheritFromParent` to `FALSE` and includes the `restoreDefault` value. The policy from the **Organization Node** is not inherited, and the default constraint behavior is used, so the effective policy evaluates to allow all.

Hierarchy evaluation rules

The following rules govern how an organization policy is evaluated at a given resource. You need the **Organization Policy Administrator** (https://cloud.google.com/resource-manager/docs/organization-policy/using-constraints#add-org-policy-admin) role to set organization policy.

No organization policy set

If you don't set an organization policy, then a resource node inherits from its parent. If the parent is the organization node or the parent node doesn't have an organization policy, then the default behavior of the constraint is enforced.

Inheritance

A resource node that has an organization policy set by default supercedes any policy set by its parent nodes in the hierarchy. However, if a resource node has set `inheritFromParent = true`, then the effective Policy of the parent resource is inherited, merged, and reconciled to evaluate the resulting effective policy. For example:

- A folder denies the value `projects/123`.

- A project underneath that folder denies the value `projects/456`.

The two policies are merged, and in this case result in an effective policy that denies both `projects/123` and `projects/456`.

Disallow inheritance

If a resource hierarchy node has a policy that includes `inheritFromParent = false`, it doesn't inherit the organization policy from its parent. Instead, the node inherits the constraint's default behavior unless you set a policy with allowed or denied values.

Reconciling policy conflicts

When a child node inherits organization policies based on list constraints (https://cloud.google.com/resource-manager/docs/organization-policy/using-constraints#list-constraint), the inherited policies are merged and reconciled with the node's organization policy. In list policy evaluation, `DENY` values always take precedence. For example:

- A folder denies the value `projects/123`.
- A project underneath that folder allows the value `projects/123`.

The policies are merged and the `DENY` value takes precedence. The effective policy denies all values, and will evaluate the same way whether the parent or child node denies the value. It's best not to include a value in both the allowed and denied lists. Doing so can make it harder to understand your policies.

Organization policies that are derived from boolean constraints (https://cloud.google.com/resource-manager/docs/organization-policy/using-constraints#boolean-constraint) do not merge and reconcile policies. If a policy is specified on a resource node, that `TRUE` or `FALSE` value is used to determine the effective policy. For example:

- A folder sets `enforced: true` for `constraints/compute.disableSerialPortAccess`.
- A project underneath that folder sets `enforced: false` for `constraints/compute.disableSerialPortAccess`.

The `enforced: true` value set on the folder is ignored because `enforced: false` is defined on the project itself. The organization policy will not enforce the constraint for that project.

Reset to default policy

By invoking `RestoreDefault`, the organization policy will use the default behavior of the constraint for this resource hierarchy node. Child nodes will also inherit this behavior.
