# Resource Policy Blog Series – 1

I have been talking with customers on IT Policies for their Azure resources recently. I would like to summary some common patterns I observed so far and how to use Resource Policy to solve the issues.

## Scenario 1 : Enforce tag in order to monitor resource usage

Let’s take a typical example. In an enterprise, there’re lots of users using Azure subscriptions. IT admins from the infrastructure team are owners of the subscriptions, and IT guys from other different departments are contributors.

For track internal costing, IT admin are using Tags to organize their resources ( one way to do it is documented [here](https://azure.microsoft.com/en-us/documentation/articles/resource-group-using-tags/)). Also, IT admin can use Tags for other purposes, e.g an external management service requires specific tags to work properly. In order to make sure Tags appears in their resources, very often, it is a manual process or an automated process done periodically to add tags according to certain rules. Even with automation, there is no confidence that required tags are neatly attached to every resource. In short, this requires lots of work and attention and can sometimes go wrong. Therefore, a desire to enforce tags on Azure resources is appreciated. The new feature of Resource Policy in Azure Resource Manager has been designed to tackle this scenario. And here is how.

To rephrase the requirement, the general policy is that every Azure Resource must be associated with a tag that contains department key and with a valid value. Similar to the documentation [here](https://azure.microsoft.com/en-us/documentation/articles/resource-manager-policy/), I can have a policy looks like below:

{

"if": {

"not": {

"allof": [

{

"field": "tags",

"containsKey": "department"

},

{

"field": "tags.department",

"in": [ "department1", "department2", "department3" ]

}

]

}

},

"then": {

"effect": "deny"

}

}

There could be a few variations to this statement, such as the policy for just for VM resources. With the recent support of nested conditions, you can add a resource type condition. Then your policy would look like below.

{

"if": {

"allOf": [

{

"not": {

"allof": [

{

"field": "tags",

"containsKey": "department"

},

{

"field": "tags.department",

"in": [ "department1", "department2", "department3"]

}

]

}

},

{

"source": "action",

"like": "Microsoft.Compute/\*"

}

]

},

"then": {

"effect": "deny"

}

}

From the standpoint of authoring, you would always want to define your policy that works against the specific condition you want, since the policy definition is a deny policy in nature. When you have multiple policies, all of them will be evaluated and any of them will lead to the deny effect if the condition is met. Therefore, you can have a general policy saying every resource must have a department tag, and every resource under Microsoft.Compute must have one additional application tag. Things you can’t do is a general one says every resource should have a department tag, but VMs can be exception. The goal is to have least interferences among policies, so that you don’t have to change existing policies when adding new ones.

## Scenario 2: Extension of Access control

Well, having tag enforced is great. Another thing usually unbearable is allowing creation of arbitrary resources. One way to do it is through RBAC. You can create a role with the permission to the service whitelist. This is the “VM admin”, “Website Admin” you generally see and the right way to deal with this problem. Then what if you have the following scenarios:

* Resources of particular SKU are not allowed
* for a particular resource group with my production workloads no reboot allowed

For scenario #1, there is no way you can do it today. RBAC is not a choice since it only looks at actions. However, ARM is adding support in policy languages to properties in the property bag, so that you can block creation of resources of specific kinds. Initially, only a small subset of properties are supported.

{

"field": "Microsoft.Storage/storageAccounts/accountType",

"in": ["LRS\_Standard", "GRS\_Standard", "ZRS\_Standard"]

}

For scenario #2, of course, you can create a special role of ProductionWebAdmin. However, a nicer way is to still use WebAdmin but create a Policy to deny restart operations on VMs. Remember, RBAC gives you permission and Policy denies. I’ll let you do an exercise of comparing the pros and cons. (This is again not supported yet, since Policy today only evaluate PUT requests, but it should be added in near future). The policy definition will look like below:

{

"if": {

{

"source": "action",

"like": "Microsoft.Compute/virtualMachines/restart"

}

},

"then": {

"effect": "deny"

}

}

Yes, you may have now found Policy is about enforcement by denial. Other interesting you can do with Policy:

* restrict resource creation to specific types
* restrict resource creation to specific locations
* restrict role assignments to only specific roles

In my next blog, I’ll talk about how to use Resource Policy for finer granularity control over resources.