This Terraform code defines several Azure resources related to Azure Automation. Here's a breakdown of the resources created:

1. **Azure Automation Account**: This resource represents an Azure Automation account. Azure Automation is a cloud-based automation and configuration service that provides consistent management across your Azure and non-Azure environments. The automation account is where you store all of your runbooks, configurations, and other automation-related assets.
2. **Azure Role Assignment**: This resource assigns a role (in this case, "Contributor") to the automation account. This role assignment grants permissions to manage Azure resources.
3. **Azure Automation Runbook**: Runbooks are a set of tasks that can be executed in Azure Automation. This resource defines a runbook that contains automation scripts or workflows. Runbooks can be run manually, on a schedule, or triggered by other Azure services.
4. **Azure Automation Schedule**: This resource defines a schedule for running the automation runbooks. It specifies when and how often the runbooks should be executed. Schedules can be one-time, recurring daily, weekly, or monthly.
5. **Azure Automation Job Schedule**: This resource ties together the runbooks and schedules, creating job schedules. It specifies which runbook should be executed according to which schedule and provides any necessary parameters for the runbook.

The actual use of Azure Automation can vary widely depending on the specific automation needs of an organization. Some common use cases include:

* **Automating repetitive tasks**: Azure Automation can be used to automate routine and repetitive tasks such as VM provisioning, backup management, and patching.
* **Configuration management**: It can be used to ensure consistent configuration across Azure resources and on-premises systems.
* **Process automation**: Azure Automation enables the automation of complex processes and workflows, helping organizations improve efficiency and reduce manual errors.
* **Monitoring and remediation**: It can be used to monitor resources for specific conditions and trigger automated remediation actions when issues are detected.
* **Integration with other Azure services**: Azure Automation can integrate with other Azure services such as Azure Monitor, Azure Log Analytics, and Azure Functions to create comprehensive automation solutions.

Overall, Azure Automation provides a powerful platform for streamlining operations, improving reliability, and reducing operational costs in Azure environments.

**Ability to Start Stop AKS Clusters from github actions**

[Edit](https://github.com/LexisNexis-RBA/bussvc-sre/wiki/Ability-to-Start---Stop-AKS-Clusters-from-github-actions/_edit) [New page](https://github.com/LexisNexis-RBA/bussvc-sre/wiki/_new)

Raghu edited this page on Feb 5 · [7 revisions](https://github.com/LexisNexis-RBA/bussvc-sre/wiki/Ability-to-Start---Stop-AKS-Clusters-from-github-actions/_history)

The ability to Start / Stop the AKS Clusters from github actions is a very nice option to have. This can help is reducing costs. This gives the development team also the flexibility to stop/start their AKS Clusters due to which we can automatically shutdown the clusters even earlier. In addition to this in cases of blue / green swap, the development team can themselves bring up the AKS clusters without us having to use the breakglass option or the SREs being in the flow.

With this in place, you will get actions like below in your app repository.

What this essentially boils down is the ability to access the aks clusters from the github actions. So this gives us more flexibility is running many automated tasks like verifying the expiry of the ssl certificates etc. Below are the sequence of steps needed to get this up and going in your applications.

**Step 1 - Open a policy exception for your dev/nonprod/prod subscription**

We need the Federated Credentials with Managed Identy policy exception to be applied on our subscriptions. So Open an issue in the policy github repository like [this](https://github.com/LexisNexis-RBA/terraform-azurerm-policy/issues/1195) for each of your dev / nonprod and prod subscriptions

**Step 2 - Create Federated Credentials with Managed Identities**

What this essentially means is

* Create a Managed Identity in Azure in that subscription
* Allow it access to work on the resource group containing the aks cluster
* Mention From which Github repository can this Managed Identity access Azure
* Mention the environment/branch of the Github repository from which this Managed Identity can access Azure

Let us see how to do this.

* First create a new directory in your iac workspace called misc/blue
* Add the standard things like configuring the backend.tf, data.tf etc
* In main.tf, add code like below

module "identities" {

source = "git@github.com:LexisNexis-RBA/terraform-bsvc.identities?ref=v1.0.0"

location = data.terraform\_remote\_state.metavnet.outputs.location

resource\_group\_name = data.terraform\_remote\_state.metavnet.outputs.resource\_group.name

tags = data.terraform\_remote\_state.metavnet.outputs.tags

user\_identities = local.user\_identities

aks\_user = local.aks\_user

}

* In locals.tf add code like below for the user\_identities and the aks\_user vars

"aks\_mi\_user" : {

"roles" : {

"role1" : {

"scope" : "/subscriptions/${local.subscription\_id}/resourceGroups/${data.terraform\_remote\_state.metavnet.outputs.resource\_group.name}"

"role\_definition\_name" : "Reader"

},

"role2" : {

"scope" : "/subscriptions/${local.subscription\_id}/resourceGroups/${data.terraform\_remote\_state.metavnet.outputs.resource\_group.name}"

"role\_definition\_name" : "Contributor"

}

},

"federated\_identity\_creds" : {

"ffred\_cred1" : "repo:LexisNexis-RBA/bweb.fedexfred:environment:dev"

"ups\_cred1" : "repo:LexisNexis-RBA/bweb.ups:environment:dev"

"ffast\_cred1" : "repo:LexisNexis-RBA/bweb.fedexfast:environment:dev"

"fat\_cred1" : "repo:LexisNexis-RBA/bweb.fedex\_admin\_tool :environment:dev"

"mfa\_cred1" : "repo:LexisNexis-RBA/bweb.multi\_factor\_auth:environment:dev"

"rest\_cred1" : "repo:LexisNexis-RBA/bweb.rest\_server:environment:dev"

"opt\_cred1" : "repo:LexisNexis-RBA/bweb.opt-out:environment:dev"

}

}

}

aks\_user = "aks\_mi\_user"

* The aks\_user var indicates the name of the managed identity
* The aks\_mi\_user variable has 2 sections

**roles** section where we are giving it contributor and reader permissions to the resource\_group where the aks cluster lives  
**federated\_identity\_creds** section, which is telling that this managed identity will be able to hit azure from what all github repository and particularly which environment of that github repository.

In the above example which was bsweb subscription, its a subscription where mulitple apps run, so we had listed multiple github repos In your case, it might be just one github repository where the main application code resides. Also the keen observer would have noted that we have given 2 roles to this managed identity. However when it comes to production, we will give it 4 roles, 2 for operating the blue cluster and 2 for operating the green cluster.

Below are the links to working example in diff environments

1. [dev](https://github.com/lexisnexis-iac/businesssvc-bsweb-dev/blob/main/misc/blue/locals.tf)
2. [nonprod](https://github.com/lexisnexis-iac/businesssvc-bsweb-np/blob/main/misc/blue/locals.tf)
3. [prod](https://github.com/lexisnexis-iac/businesssvc-bsweb-pr/blob/main/misc/blue/locals.tf)

Once you apply the above changes, you have all the things in place from the Azure Side. i.e

* You have created a manged identity which has access to the AKS cluster
* You have mentioned the list of github repos where you will have github actions to work on the aks cluster
* You have mentioend the list of environments within github where the aks-cluster can be worked on.
* In the prod version, there is an environment called **checks** in addition to **prod**, where we will run some automated checks
* so are all done on the Azure side. Now we will go to the github side

**Step 3 Create Github Environments in the app repo**

Here we will go with the example of the bweb.ups repo, which is the application repo for this example.  
Go to Settings -> Environments and hit the button to create 4 new environments

1. dev
2. nonprod
3. prod
4. checks

It will look like below

Now click on each environment and add 3 Environment secrets within them, namely

1. TENANT\_ID
2. SUBSCRIPTION\_ID
3. AZURE\_CLIENT\_ID

The first 2 you are familiar with wrt your subscription. For the **AZURE\_CLIENT\_ID** that will be the Client id of the managed identity you created via TF. Go to the managed identity from the azure portal to get the value of the Client ID of the Managed Identity. Note that only the tenant\_id value will be same across the 4 different environments in which you are adding this environment secrets. The other 2 values will change in different environments. However the secret values will be same in the **prod** and the **checks** github environments. They will be looking like below

In github prod environment, you alread my have **Deployment protection rule**. If not, discuss with your app team owner and create one. What this does is, when we apply the action on the prod environment, it waits for others to approve before the action is executed. So in our case, if we want the bring down the aks blue cluster, then after the action is triggered, it will wait for confirmation from others in that app before actuall doing it. It will look like below

**Step 4 Bring in the actual github actions to do the task**

These are some of the actions that we can have

* aks-start action
* aks-stop action
* argo-rollouts-actions
* ssl\_cert\_expiry\_check action

Copy the below files onto your applications .github/workflows directory

<https://github.com/LexisNexis-RBA/bweb.ups/blob/develop/.github/workflows/aks-start.yaml>

<https://github.com/LexisNexis-RBA/bweb.ups/blob/develop/.github/workflows/aks-stop.yaml>

<https://github.com/LexisNexis-RBA/bweb.ups/blob/develop/.github/workflows/ssl_cert_expiry_check.yaml>

<https://github.com/LexisNexis-RBA/bweb.ups/blob/develop/.github/workflows/argo-rollouts-actions.yaml>

In the aks-start and the aks-stop actions, you can see the sections where we are listing out the aks cluster name and its associated resource group for each environment like dev/nonprod/staging/blue/green. You have to change them accordingly in your case. from below section

In the other 2 actions, i.e. the ssl-cert-check action and the argo-rollouts actions, over here too change the cluster name and the resource group [ most likely to only your prod versions ] as below

**Step 5 - Test it out**

Commit all the workflow files into the github repo and go to the github actions and start testing the various actions against all the environments.

*The common reusable actions are present over*[*here*](https://github.com/LexisNexis-RBA/bussvc-actions/tree/main/.github/workflows)*in case you are curious on how it works under the hood. If you have additional ideas on common - reusable github actions that can help managing our azure infra reach out to me and we can use them.*