# GitHub ARM IPKit

For the class we are supplying you with an Infrastructure as Code system leveraging GitHub Actions and the <https://dev.azure.com/servicescode/infra-as-code-source> modules and environments. We’ll go over these instructions and then you’ll go back to your focus room and complete them in your group. You may want to choose one individual to drive with a screen share and work together to complete them.

Two methods used to deploy ARM GitHub workflows are shown below. For this class I would recommend using the first as it works better with the IaC Modules we will use:

* Custom GitHub Actions from the DevOps Geek Garage community that:
  + Run from a self-hosted GitHub runner (i.e. build agent) inside a docker container
  + Can leverage the VM’s Azure system managed identity providing password-less authentication
* Marketplace GitHub Actions that:
  + Runs in either hosted or self-hosted runner using service principal secret stored in GitHub
  + Runs powershell and ARM as commands
  + Specific ARM GitHub Action has an issue with some outputs from the IaC Modules ARM templates we plan to use

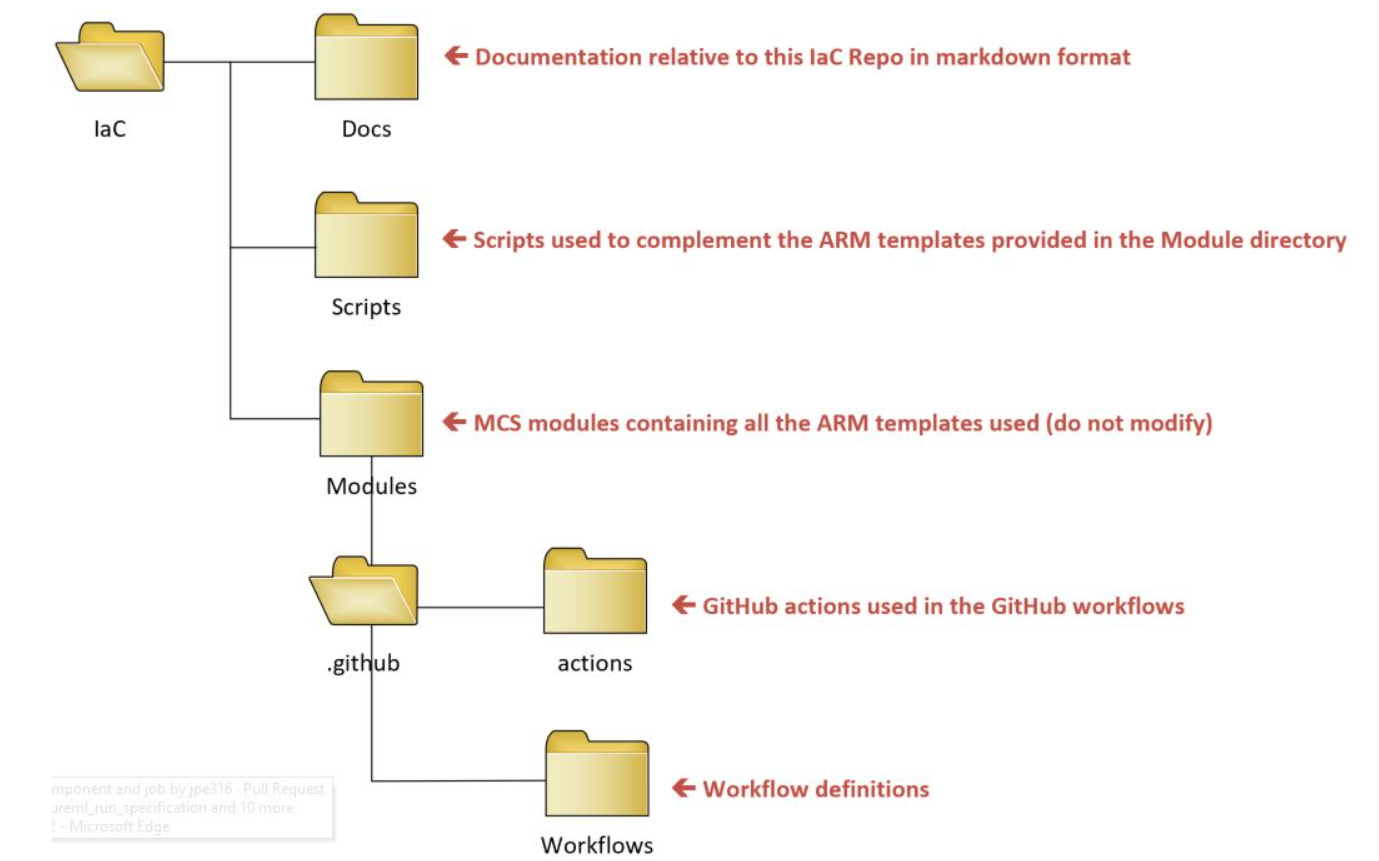
You can access the repository [here](https://github.com/jriekse5555/FedACFIaC). You should have submitted your github handle ahead of the class so you can accept the invite for permissions to the repo. If you don’t have access please let the proctor team know.

The first step is to clone the previously referenced repo. This can be done by one or more of your group members.

## **Setting up the new IaC repo for your project**

Follow [these instructions](https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-from-a-template) to create a new repository based on the template for this IaC repository. When you do make sure you set the repo visibility to PRIVATE as this code base is not public.

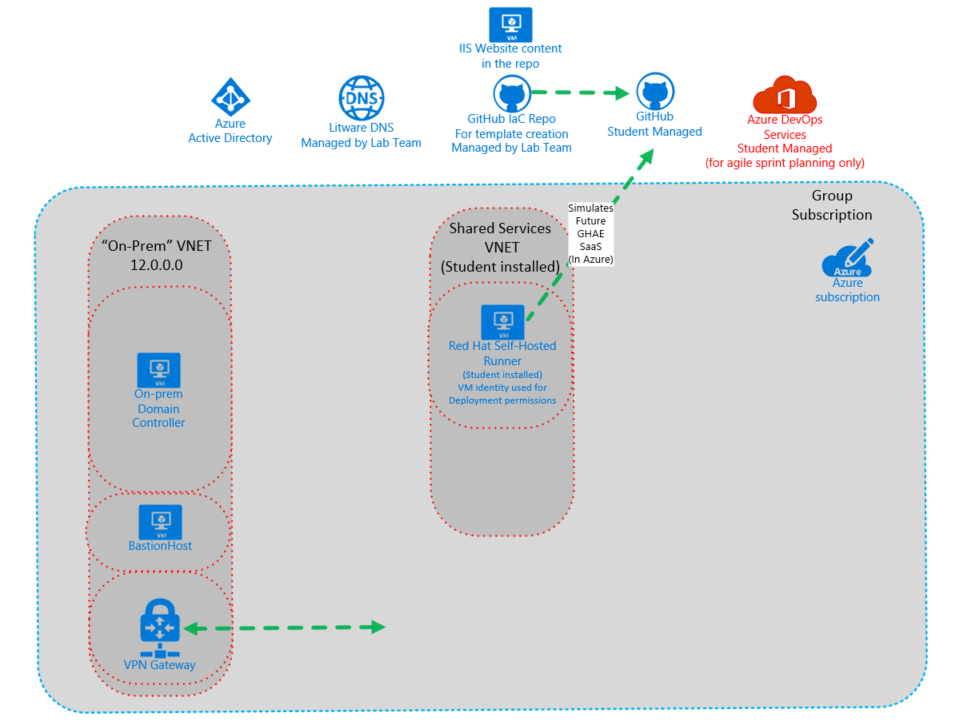
The new repository structure should look like the following:



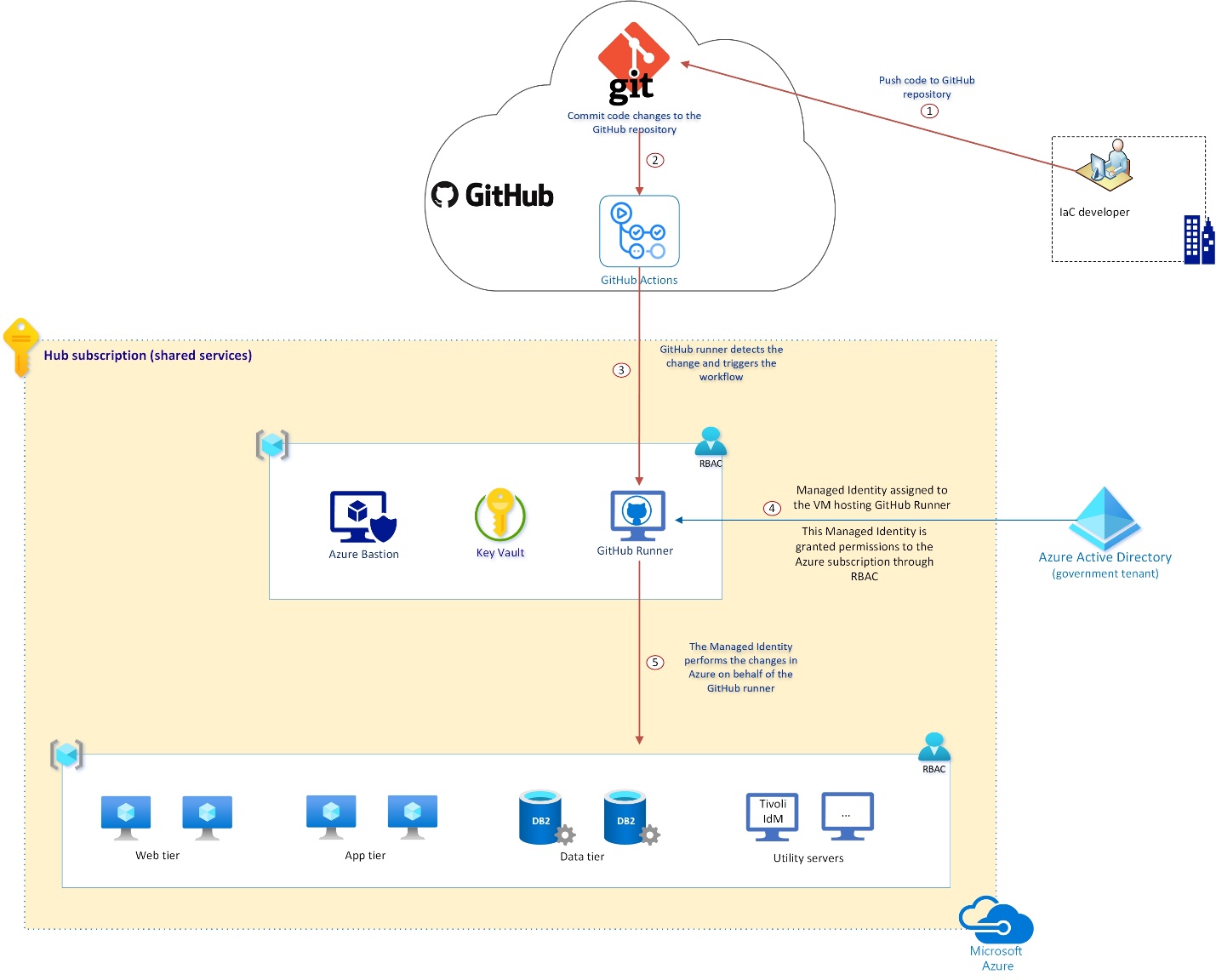
* The **Docs** directory contains this documentation
* The **Scripts** directory contains no ARM based code used to deploy or configure resources in Azure.
* The **Modules** directory contains all the ARM templates used in this project. There is one module per Azure resource type.
* The **.github** directory contains the github workflows developped for the project to deploy Azure resources. These workflows are stored in the **workfow** sub-directory and they call the GitHub actions located in the **actions** sub-directory.

To support the method mentioned above next setup an Azure Red Hat VM to be used as the self-hosted runner. You’ll also need to ensure the network path is clear to connect to the GitHub commercial site. Once this is setup and registered in GitHub you’ll be able to use the VM managed system identity to provide the rights for deployment. Thanks to the MCS GitHub Incubation and DevOps Geek Garage communities for the system we’ll use in the class.

See the suggested location below to install the Red Hat self-hosted runner in the Azure environment. The recommended location is in a shared services vnet and the github self-hosted runner would reside a specific subnet that for this class will allow by exception internet access to GitHub commercial. This subnet is meant to simulate a subnet designated for the upcoming [GHAE](https://github.com/github/roadmap/issues/53) Azure FedRAMP High compliant SaaS solution. With this new service this subnet would connect to this Azure service instead of to GitHub cloud as we are doing for this class. The shared services vnet should be incorporated into your larger design and can be installed manually to get your automation system ready to be used. Note to consider your IP ranges for this vnet as part of a larger scheme.



Also, see the diagram below that shows how the overall system will work from a github perspective.



# **IaC Repository**

# **Setting up self-hosted runner**

[Self-hosted runners](https://docs.github.com/en/actions/hosting-your-own-runners/about-self-hosted-runners) provide many benefits, among which the ability to prevent the storage of secrets within the GitHub Cloud. When using a self-hosted runner, you can configure your workflows to run under a managed identity instead of a Service Principal whose password has to be storted in the GitHub Cloud.

The diagram below depicts how a GitHub runner hosted in your Azure subscription would work:

The instructions below provide the steps for installing the runner on Linux RedHat. You can also install runners on Windows and MacOS.

## **Create self-hosted runner on Redhat**

1. Create a VM running RedHat 7.7 or higher (7.8 gallery image preferred)

VM spec recommendations are a small B2s sku, RHEL 7.8 gallery image (Red Hat Enterprise Linux 7.8 – Gen1), public IP and the default amount of disk space on the OS drive. Without an Azure public IP the VM will not be allowed to connect to the Azure RHUI update repos which is needed to set it up. Feel free to provision the VM in the way you are most comfortable including the portal. From the diagram of the lab environment shown previously, its recommended to create a shared services vnet (assuming your component architecture is using vnets and not subnets), create a GitHub subnet, and then provision the VM there. In this lab note that the GitHub subnet is allowed to connect outbound directly to the internet to GitHub commercial to make the scenario work correctly. If you need to connect to the VM via SSH you’ll need to have a SSH tool to connect like Putty. The expectation is that if you may allow inbound connections with a public IP to get this setup, but you’ll remove them when Azure Bastion is setup.

1. The following shell [script](https://github.com/jriekse5555/FedACFIaC/blob/master/Scripts/RHEL7xSelfHostedRunnerShell/Bash_script_for_self_hosted_GitHub_runner_RHEL_v7.sh) has been created to automate the configuration of the VM. In your cloned repo it will be in the Scripts/RHEL7xSelfHostedRunnerShell sub-directory. This script takes two parameters (repo URL and token) that are shown below:’

########-------- Parameters --------#########################################

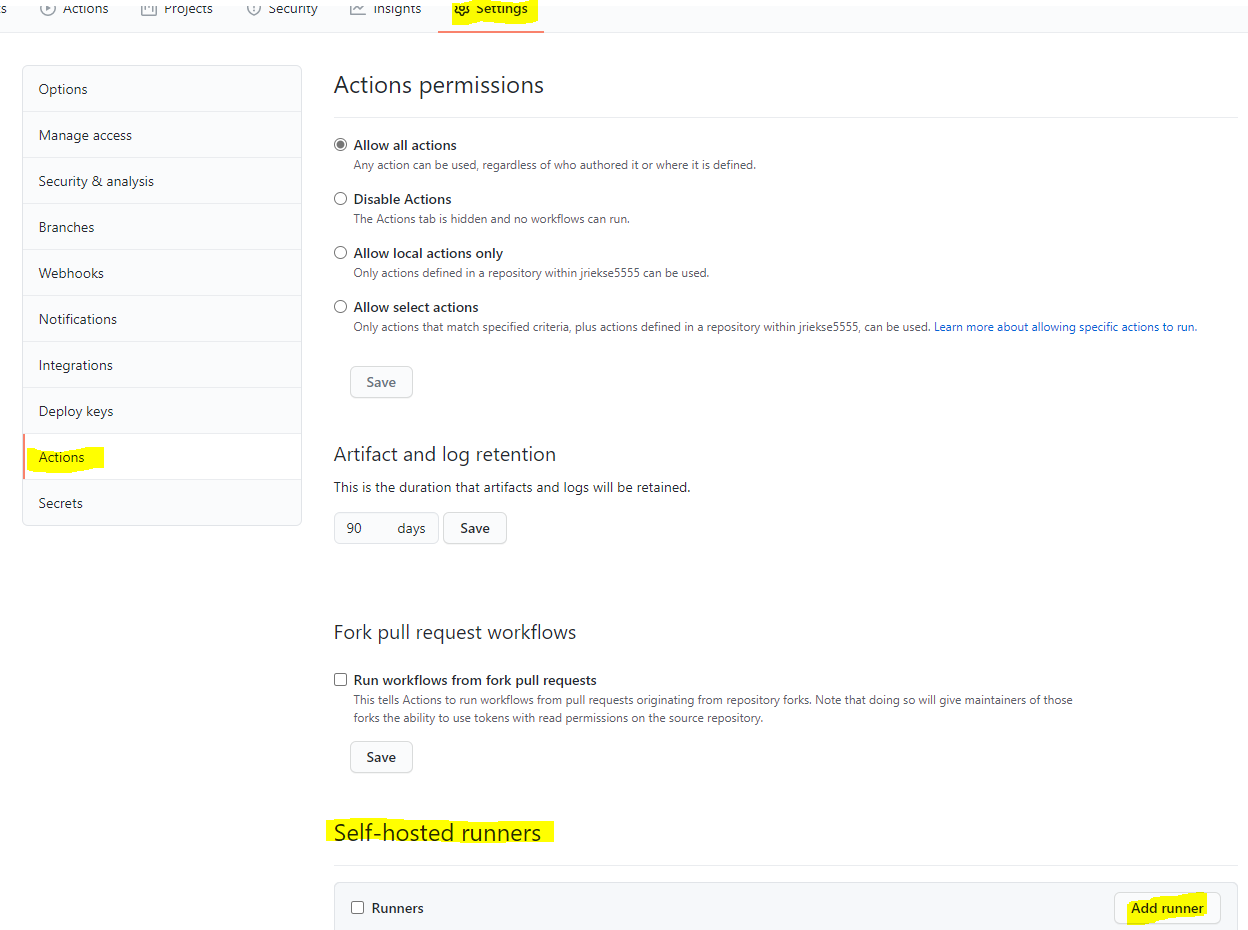
# $1 The path for the Github repo, https://github.com/<orgname>/<reponame>#

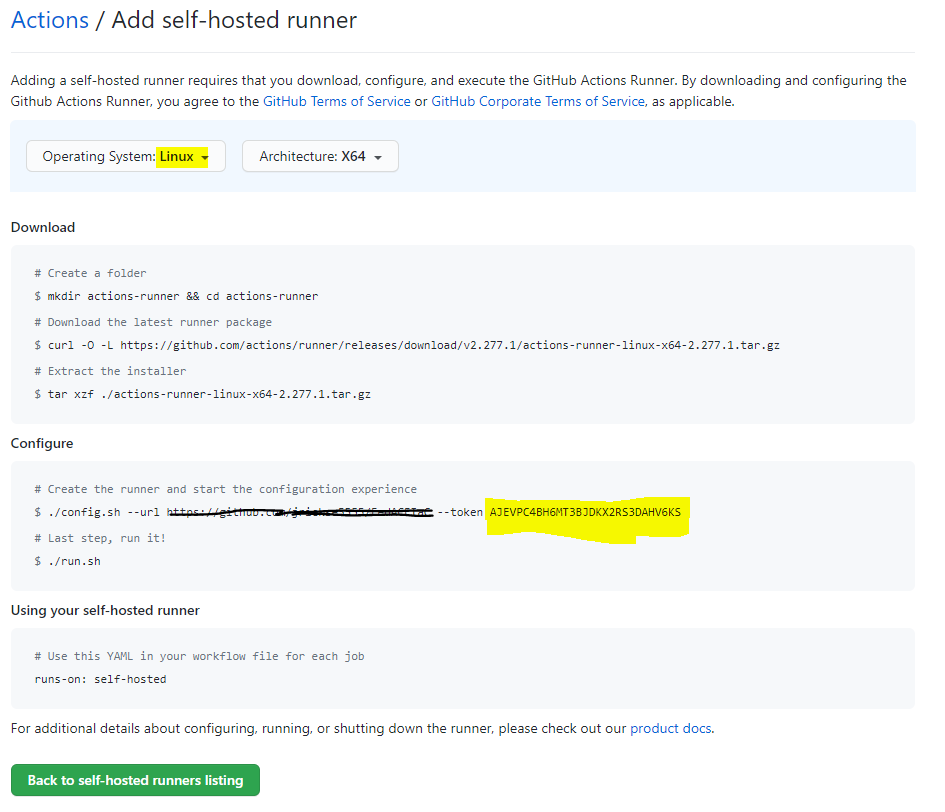
# $2 Github Account PAT Token with access to the desired repo             #

##########################################

Go to your group’s repo for the main URL which should be in the format: https://github.com/<org>/<reponame>

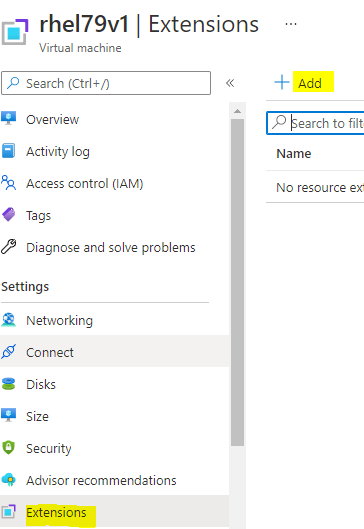
The token is found in your group’s GitHub repo under Settings, then Actions, then Self-Hosted Runner, then Add Runner:



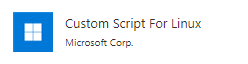


The script can be run several ways:

1. Use the VM Custom Script Extension to run the script (Note the script takes at least ten minutes and you don’t have much visibility to its status using this method)
   1. Create a storage account and upload the script there
   2. Add an extension

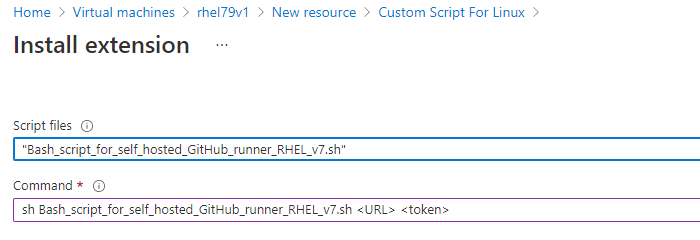


* 1. Add the custom script extension





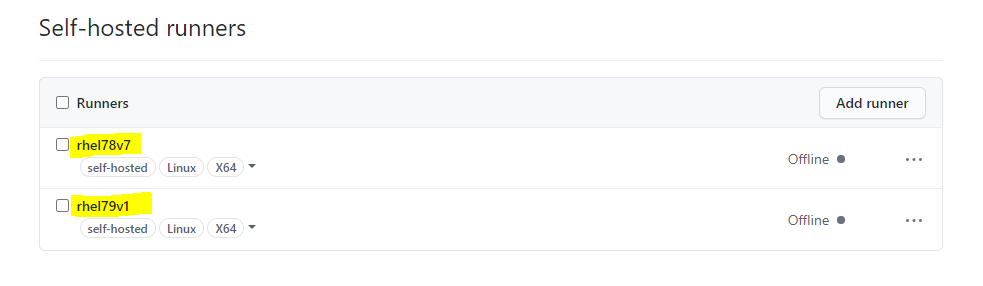
* 1. Browse to the script and add the parameters like below:



1. Logon to the server via SSH. Then either modify the script first with the two parameters and paste it into the shell. Or copy the .sh file to the VM and run it with the parameters. You’ll have visibility to the entire operation using this method.

If your group is having issues please let the proctor’s know.

1. Verify that your self-hosted runners are registered and ready to do work



1. To setup Managed identity for this VM, use either a system or user defined managed identity. A [system managed identity](https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/qs-configure-portal-windows-vm#enable-system-assigned-managed-identity-on-an-existing-vm) is simplest, but either work.

You can refer to the document [Create a user-assigned managed identity](https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/how-to-manage-ua-identity-portal#create-a-user-assigned-managed-identity) and assign it to the GitHub Runner VM. Grant this Managed Identity the proper level of access to the subscriptions that you want it to manage. You may want to give it Contributor rights on the subscription to make your deployments easier. Finally, set an Access Policy in Key Vault to grant this Managed Identity permission to "get" secrets.

# **Using the Infrastructure as Code (IaC) System**

## Install the software needed

To be able to develop ARM templates and GitHub workflows, we recommend you install the following software on your workstation:

* [Git](https://git-scm.com/download/win)
* [Visual Studio Code](https://code.visualstudio.com/Download)

Click on the links above and install the two packages by following the instructions provided.

Then start VS code and install your favorite extensions. Some of the ones we recommend are listed below:

* [GitLens — Git supercharged](https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens)
* [Azure Resource Manager (ARM) Tools](https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools)
* [YAML](https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml)
* [Markdown All in One](https://marketplace.visualstudio.com/items?itemName=yzhang.markdown-all-in-one)

## **Sync your repo locally**

You’ll likely want to setup git on your client computer to sync with the github repo. To do so once you have created the GitHub template repo and installed Git for Windows previously:

* On your local computer, open a command prompt
* If you don’t have other git repos synced, Run:
  + git config --global user.email <username@org.com>
  + git config --global user.name "<Name>"
* If you don’t already have a directory to store git repos create one
* Navigate to this directory within the command prompt
* Run:
  + git clone [https://github.com/<yourGitHuborg>/<reponame>.git](https://github.com/%3cyourGitHuborg%3e/%3creponame%3e.git)

## **Test the system**

As a reference for later, to find out how the actions and workflows need to be setup under the .github directory, click on [Setup your Workflows](https://github.com/jriekse5555/IaC_GitHubActions/blob/master/Docs/Setup-your-workflows.md)

Here are the steps to test the system and advice for using it:

* Create a resource group for the test deployment
* Ensure the self-hosted VM identity has access to this resource group and other resources it will need. In this test, we are deploying a Log Analytics workspace and this identity will need read access across the subscription. You may want to enable the identity with Contributor at the subscription level.
* First, open the file below in Visual Studio Code or the editor of your choosing:
  + C:\<localgitlocation>\.github\workflows\rg-githubtest\LogAnalytics\parameters.json
* Edit the parameters file with a name of a log analytics workspace that will be unique across Azure
* Save the file
* Open the .yml file on:
  + C:\<localgitlocation>\.github\workflows\rg-githubtest.yml
* Edit the following and save the change
  + Update the resourceGroupName to the resource group created earlier
* Note that once this update is made the GitHub workflow will start running. There is no manual start at this time so the on: section of the .yml will determine what changes start the workflow.
* Add the commits and push the changes to the repo
* You can check the status of the GitHub workflow in the Actions tab in GitHub

IMPORTANT: Additional .yml templates are in the repo that give examples that may be very helpful. However, note that many of the module paths need to be corrected to remove the version in the path as it doesn’t align to where the modules actually are and the on: section needs to be set correctly to ensure the pipeline starts when changes are made.