GitHub Governance Demo Script

# For GitHub on EXP

## August 15, 2020

### Purpose

The goal of this demo is to show how you can use GitHub to set up project code repositories and CI/CD pipelines in a manner to afford correct governance for an Enterprise. The general CI/CD guidance should follow Microsoft best practices. A brief summary of which is:

* 4 environments: Dev, Test, Pre-Prod, Prod
* IaC deployment from Dev to Test
* Blueprints for Pre-Prod and Prod
* Policies for all 4 environments, but not deployed as part of the code pipelines, as you want them in place before the code is deployed.

Due to the state of GitHub in 2020, at some point in the engagement discussion, it may become obvious that the best practice is to integrate with Azure DevOps and continue on in that system. Integration with Azure DevOps is both valid and desirable in many cases, but won’t be covered in this Demo Script. We will look to update this script as GitHub matures over the next 12-24 months.

### Audience

Enterprise customers who are moving into Azure (or moving to set up governance in Azure) and are either trying to set up CI/CD for the first time, or adding Governance to their existing ad hoc processes.

This script is primarily targeting:

* Organizations who have been using Azure DevOps/TFS and are trying to see how GitHub compares from the standpoint of governance and Microsoft best practices. The goal is for these organizations to understand the governance story in GitHub, even if they are not going to adopt GitHub for the next 12-60 months. They should understand that eventually, they will be able to have governance in GitHub as they do in Azure DevOps. **It is important to be upfront with Azure DevOps customers that the governance story is currently much better in Azure DevOps than GitHub. Customers with a governance focus will want to stay on Azure DevOps at least through 2020 (with a wait and see on how GitHub looks in 2021)**
* Organizations who have already adopted GitHub, but with a weak governance plan. We want to try to help these organizations develop better workflows.

The script is not specifically targeting:

* Organizations who are not using any of our products and are looking for reassurance that Microsoft products can provide the correct level of governance for their enterprise. These organizations should receive a combined GitHub and Azure DevOps demo. In the short term, adoption of both products would be extremely valuable, as items like Advanced Security are only available on the GitHub side, but if they cannot be convinced to go that route, they should be targeting Azure DevOps
* Organizations who are just looking for governance information around Azure (e.g. Policies, Blueprints). These organizations should receive an Azure governance briefing, which is distinct from either Azure DevOps or GitHub.

### The Script Topics

* Setting up AAD access for your GitHub Organization, using best in class identity management
  + Conditional access
  + B2B access
* Organizational control over repos, how to keep everyone from running wild
  + Restricting public/private repos
  + Enforcing notifications to a domain
  + Restricting use of 3rd party apps
  + How to be notified on Repo/Fork notification (and then take any action you want…)
  + GitHub organizational policies pushed down to individual projects/repos
  + Note that Paul has created a webhook demo wo make administrative changes. This is how the MSFT org is regulated. We can modify the demo to do more substantive things
* Integration with Azure Monitor
  + Export of Audit logs to Azure Log Analytics (may have to wait for Organizational Actions to be released, TBD, but can be done with a logic app, however, we can't get our records to show)
* IaC Dev -> Test deployment with Actions, using ARM templates. This could be a duplicate from the general workflow script we are creating. It's the same demo, but with a different talk track. Copy if possible, for consistency
  + This should include using the Azure Key Vault
* Test -> PreProd -> Prod Code Deployment with Actions. This is into blueprinted environments. Note that this is where the combined demo would switch to Azure Pipelines and release management with gates and approvals. We'll demo it the best we can to "stay in GitHub", but the reality for the short term is that this is where the combination of products demos way better.
  + This should include using the Azure Key Vault
* Blueprint deployment Test -> PreProd -> Prod. Note **not** from Actions or GH. This is an Azure piece which we need to bring in from the Azure Governance discussion, so the context of CI/CD make sense. We'll put together a 'thin' demo for this script (or pre-load them)
* Project management
  + PR management, restrictions
  + Branch policies
  + Test results using actions (status checks)
* Setting up a self-hosted runner
* Actions for governance
  + Reported content. Remove a user from access on a report being made

### The Script Story

The Contoso Organization needs having some DevOps challenges. Six months ago, they signed up for a DevOps service to hold their source code and put together their builds, but have found that their developers have created a plethora of repositories with source code. This is causing problems on merging branches, but beyond that, the company does not know how many repos exist. Some of them may not be fully locked down, and if the individual developers leave, they do not even know how to find all the pieces. They need to:

* Receive notifications to their back-end systems when repos are created in the company name or when repos are forked from existing company repos. They want to be able to control when repos are made public
* Some of the developers have been logging in with the DevOps system accounts. They would like everyone to use their AAD credentials so they can be guaranteed that identity management items, such as MFA and Conditional Access are utilized.
* They would like to receive audit records of all actions in their repos

Once that is all set, they need to have a regular development workflow for their 200 developer teams. They want a repeatable workflow from dev into production so when individual developers move teams, the general company workflow is consistent. The company wide development manager wants this to include testing and code reviews, as he feels that quality has been lagging since they moved to this DevOps system. He also wants to be able to automatically remove people from a repository on a complaint that they are not following company guidelines.

They have noticed that their Azure production environments are not repeatable, as people are going in and hand-modifying the environments when problems arise. In addition to this being worrisome, they believe that this is increasing their Azure costs beyond where they should be. They are also receiving new requirements regarding data residency for some of their worldwide systems and want to make sure that they are minimizing egress charges within each application.

Consoso also thinks that some of their Government projects will need more secure build servers than the DevOps service can offer, and want to host these for some projects in an Azure environment that lives behind a VPN gateway. They want to be able to restrict the use of 3rd party applications in the repos for these projects, until they can review them and get permission for their use from the Government.

We are going to show Consoso how they can improve their overall governance posture by moving to GitHub.

* Demo: controlling repos in their organization
  + Restricting public repos [steps]
  + Backend notification on new repo creation and fork notification [steps]
    - Do not allow forks into non org repos
  + GH Policies enforced on organization repos [steps]
    - Restrict 3rd party apps to only a white list [steps]
  + Permissions removed on reported bad behavior [steps]
* Demo: Enforcing identity in GH
  + Require AAD Access [AAD/Identity discussion + steps]
  + Allow approved visitors to a repository with AAD B2B [discussion only with artifacts]
* Demo: Receive GH Audit records into Azure log Analytics [steps]
* Demo: Show how to restrict the PR workflow and set up Branch policies [steps]
* Demo: CI build with test results using actions [steps, but GH must have an example of this somewhere to steal from]
* Demo: IaC deployment from an ARM template from Dev->Test using actions [Steps, copy from the other demo script, so that this is the same workflow (or very, very similar -- minimize work, but if they are not using the key vault in the other script, update both)]
* Demo: Blueprint deployment from Azure [Discussion only or incorporate an Azure demo. No GH steps for this]
* Demo: CD deployment to PreProd and Prod using Actions [Steps]
  + Once approvals are available in GH, modify the script, so this is V2
* Demo: Set up private build server in Azure behind VPN. Rerun all actions on this build server [steps]

### The Script

1. Let us show how GitHub access through AAD is setup. This is a demo, as our org has already been configured.
   1. Log into GitHub
   2. On the repo page, click on settings (req. admin read access)
   3. Show the SAML settings
      1. Enable SAML Authentication is checked
      2. The AAD Oauth sign on URL is filled in (provided by Azure)
      3. The SAML identity provider is filled in (provided by Azure (? -- check this with Stephen))
      4. The public certificate is provided (provided by Azure (? -- check this with Stephen)). Note that you specify the hashing algorithm.
   4. Test the SAML configuration
      1. Click on the Test SAML configuration button. It will give a test login. It will be successful.
   5. We do not click save here since we did not modify the page, but if we were doing this for real, we would
2. Once this is done, we can even synchronize a team from our Azure AAD
   1. Log into the Azure portal and show the group "GitHub group for AAD Integration". You can add users if you want
   2. In GitHub go to the onemtc page (<https://github.com/onemtc>) and click on Teams
   3. Click on the new team button
   4. Add a team name and description
   5. Under the select groups drop down, start typing GitHub, and the "GitHub group for AAD Integration" team will appear
   6. Leave visibility as "Visible" and click "Create team"
   7. From the team page, you can click on the number of users and you will see the users. Note that the "number of users on the page", is not always accurate, but the membership seems correct
3. Create a new repo in onemtc. Make it private. Let us add our new team
   1. From the repo page, click on settings
   2. Click on Invite Teams or People
   3. Choose the team you just created. Give the group "Maintain" access
4. Now that the correct folks can access the repo, let us see how we can some organization wide policies (do not actually change any of the onemtc policies, please)
   1. On the repo page, click on settings (req. admin read access)
   2. Click on member privileges
   3. Walk through the page, showing the items that can be controlled at an organization level
   4. Note that repo-level items, such as setting branch policies cannot be set at the organizational level. The organization will have to develop a policy for repos, and then it will be up to the individual repo owners to set the policies individually. (ADO works the same way).
5. Let us set up our repo to require pull requests in order to modify the master branch
   1. Go to our repo page, and click on "Settings"
   2. Click on "Branches"
   3. Click on "Add rule" in the branch protection setting
   4. In the branch name pattern box, type "master"
   5. Check only the "Require pull request reviews before merging" and "Include administrators", but walk through all the possible choices
   6. Click "save changes"
   7. If you now try to push a change to master, you will receive a rejected message
6. There are two ways to restrict public repositories. The first way is to disable the ability for organizations to create public repositories. The organization owners will still be able to create public repos. This way, security is guaranteed. This was already shown in step 4, on the member privileges page. For large organizations, this may not be the best course of action. The second way is to allow users to create the public repo, have it get created in a non-public state, but to then have an approver switch it to public after approval.
   1. Create a public repo, but the start of the name must be "onemtc-approv-"
   2. Show how this is changed automatically to private.
   3. ToDo generate an email that allows deletion of the repo. The talk track is that you could read a database and automatically allow it to be public if it was preapproved.