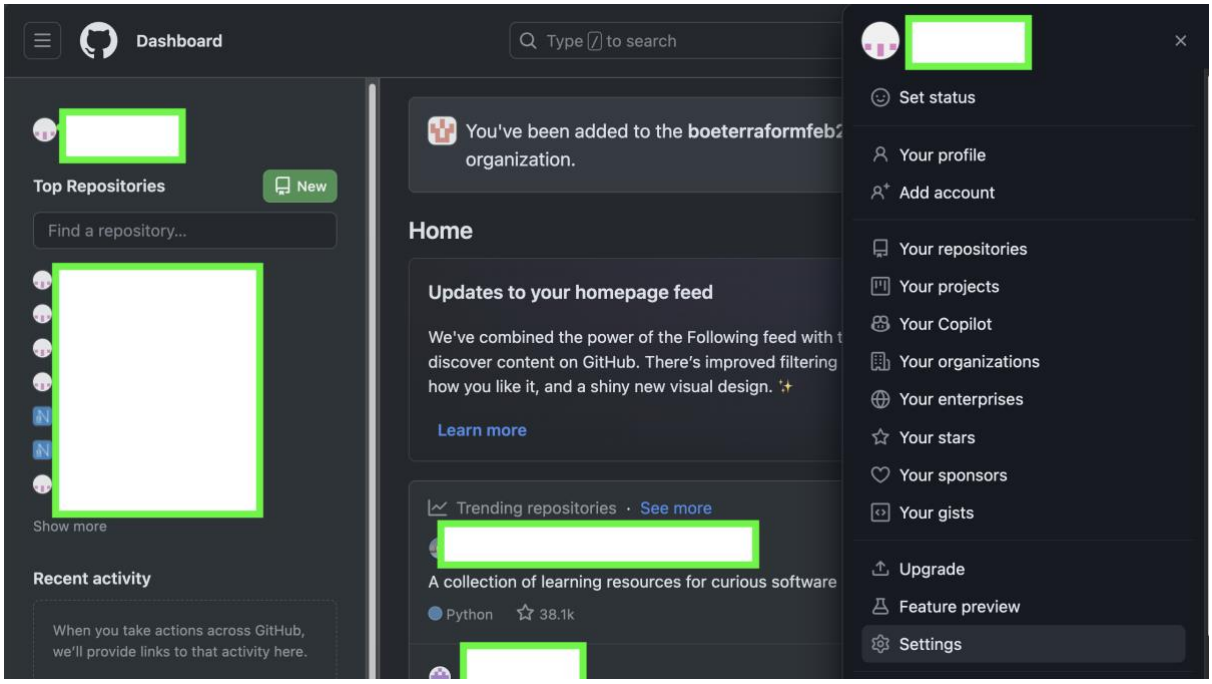
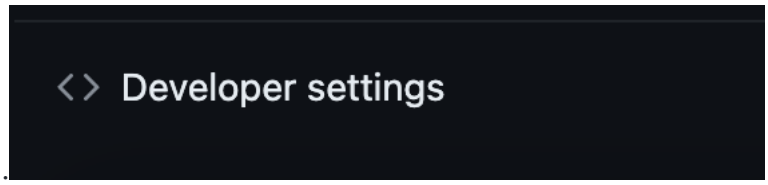
Terraform on Azure – Lab 3

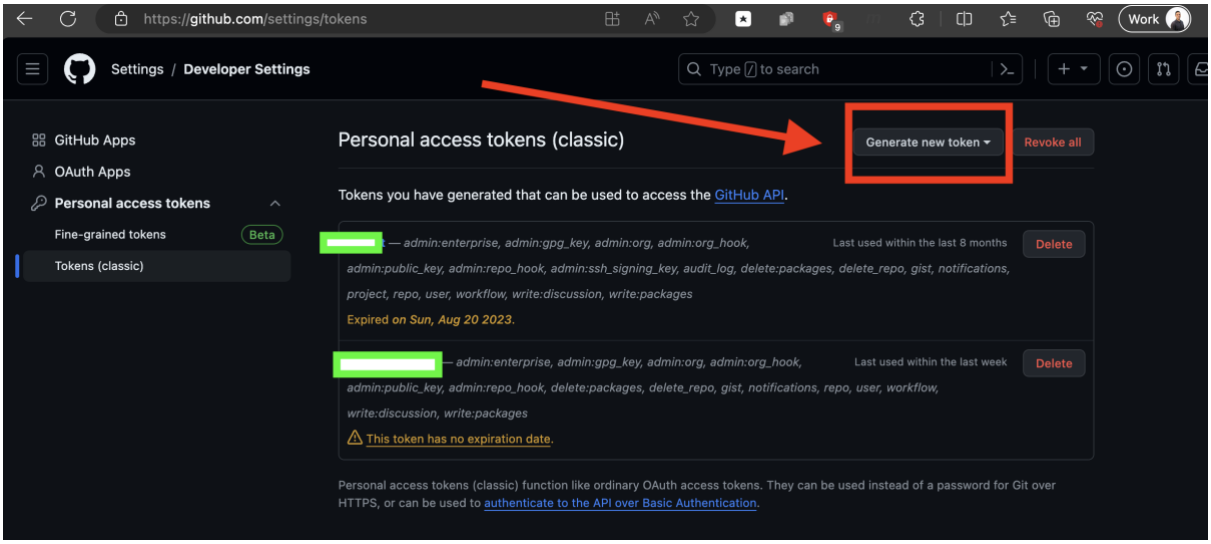
Since you are working on Infrastructure as a Code, you might want to version that code. In this lab you will version your Terraform configuration using GitHub.

1. Login to GitHub using your personal account. Once logged in go to the upper right corner (as shown on image)

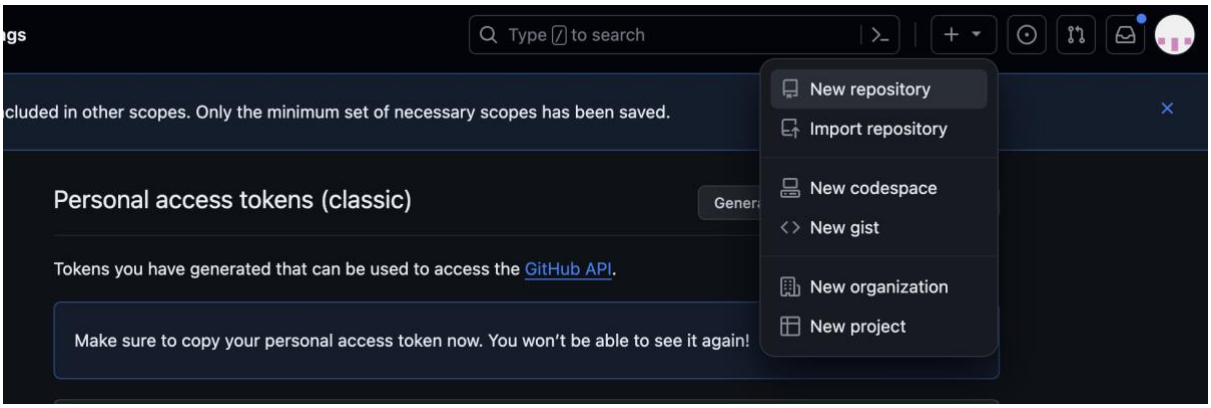


2. Click on Settings and once a new screen is presented, scroll to the right bottom of the screen and click on Developer Settings option. On newly presented screen, choose Personal access tokens, and create choose *Generate new token classic* option





3. On screen with permissions needed to be set, you can tick all the boxes but codespace, copilot and enterprise. On the bottom of the screen choose *Generate token* option.
4. You will be shown a Personal Access Token (PAT) and make sure to write down the value.
5. To create a new repo go on the upper right part of the screen and choose the option to create a new repository (as shown in the screenshot).



6. A name of your repo should be *terraformboe* . Once created, keep the windows with repo opened since you are going to use the instructions later on.
7. Back in terminal(console) navigate to terraform folder if you are not there already
8. You want to track put all the neccessary files under source control, as well as you want to restrict some files to be tracked by source control. That's why you are going to create a .gitignore file by running the following command:

```
cd. > .gitignore
```

8. The file you have just created serves as a sort of a filter that restricts certain files to ever be pushed to GitHub. The obvious candidates are .tfvars files as well as .tfstate files. Now open .gitingore file in text or code editor and append the following values:

```
# Local .terraform directories
**/.terraform/*

# .tfstate files
*.tfstate
*.tfstate.*
terraform.tfstate
terraform.tfstate.backup

# Crash log files
crash.log
crash.*.log

# Exclude all .tfvars files, which are likely to contain sensitive data, such as
# password, private keys, and other secrets. These should not be part of version
# control as they are data points which are potentially sensitive and subject
# to change depending on the environment.
*.tfvars
*.tfvars.json

# Ignore override files as they are usually used to override resources locally and so
# are not checked in
override.tf
override.tf.json
*_override.tf
*_override.tf.json

# Include tfplan files to ignore the plan output of command: terraform plan -out=tfplan
*tfplan*

# Ignore CLI configuration files
```

```
.terraformrc
terraform.rc


.DS_Store
```

Save the .gitignore file

9. After saving the .gitignore file, typing the following command in console start tracking your exisiting project with git:

```
git init
```

10. After git being initialized, add all the files by typing the following command to add all the relevant files:

```
git add .
```

11. It is time that you commit the changes, so you type the following:

```
git commit -m "Adding Terraform to source control"
```

12. You will now want to target your newly created repository and so you will use the command:

```
git remote add origin https://github.com/YOUR_GITHUB_USERNAME/terraformboe.git
```

13. Set the main branch using the command:

```
git branch -M main
```

14. And last thing is to push the code to your GitHub repository by using the following command:

```
git push -u origin main
```

If you are not already signed in to GitHub in your terminal, you are going to be prompted to do so, just be careful when enetering the password, since it is not the actual password that you should type in, rather you should paste the GitHub token that you have created at the begining of this lab.

15. If you have issues with a lot of cache and therefore not being able to push the code changes to GitHub then use the following command:

`git filter-branch -f --index-filter 'git rm --cached -r --ignore-unmatch .terraform/'` and try to push again after that.

16. Now you can open your browser and verify that your code is on GitHub

17. Let's test working in various environments. Create a new branch called dev:

```
git checkout -b dev
```

18. Create a file named `dev_secret.tfvars` copy in the values from the secret.tfvars (`client_id, client_secret, tenant_id, subscription_id`) and append the following key-value pair;

```
env_prefix="dev"
```

19. in root folder in `variables.tf` file (where client_id, client_secret, tenant_id, subscription_id are declared)and append the following:

```
variable "env_prefix" {
 description = "An environment differentiator"
 type        = string
}
```

20. Also, in modules/resource_group folder create `variables.tf` file and append the following:

```
variable "env_prefix" {
 description = "An environment differentiator"
 type        = string
}
```

21. Don't forget to add the env_prefix in the root `main.tf` file. Under resource_group module append `env_prefix = var.env_prefix` and your resource_group module should look like this:

```
module "resource_group"{
  source="./modules/resource_group"
  env_prefix = var.env_prefix
}
```

22. Let's say that you want to remove Azure Synapse Analytics workspace, and therefore you will remove the synapse configuration in modules/analytics folder (main.tf).

23. Initialize Terraform with the following command:

```
terraform init
```

24. If everything is ok, create a plan with new .tfvars file:

```
terraform plan -var-file=dev_secret.tfvars -out lab3.tfplan
```

25. Most probably, you will get a message inside of a plan that all the resources will be destroyed and new ones will be created. You only want the new ones to be created, and that's why you won't apply that plan. Rather you are going to create a workspace. But before that, you should list all the available workspaces (and you should only have the default one):

```
terraform workspace list
```

26 To test out the new configuration, you will want to create a new workspace using the command:

```
terraform workspace new dev
```

27. Once in a new workspace, you again run the command to create a plan:

```
terraform plan -var-file=dev_secret.tfvars -out lab3.tfplan
```

28. And you will see that the new configuration is completely isolated from the previous one, and therefore you run the following command to create resources:

```
terraform apply lab3.tfplan
```

29. Satisfied with your configuration, you commit and push the code changes into the dev branch by using the following commands:

```
git add .
git commit -m "Created workspace for dev env, deleted synapse"
git push -u origin dev
```

30. Switch back to the original workspace by running the command:

```
terraform workspace select default
```

You have successfuly completed Lab 3! Congratulations!