Table of content

Web-application assignment

Tutorial with step-by-step instructions on how to deploy a simple web application, with notes on what services, tools and concepts are used and why they were chosen.

Pre-requisites: A developer-IDE with a terminal, GitHub and Azure accounts, preferably with SSH keys already set up, otherwise please refer to Module X first as it is recommended to have this already set up for safety reasons as well as easier management. Basic git knowledge, the user should know how to work with branches.

Module 1: Create the application

Start off by creating the backend application. It will be constructed in .NET, using the MVC (Model-View-Controller) pattern. Start your IDE of choice (the examples in this tutorial will use VS Code version 1.97.2).

1. Open the terminal and use the “+v” icon on the right to open up a new terminal, using the Bash language.
2. Navigate to the directory where you want your application folder to be, I recommend a shorter path near the root (I use C:/DevProj/ in a Windows environment). Within that folder, type this:

```
mkdir YourFolderName
cd YourFolderName
```

3. Now you have a folder created and navigated to it. Let us start the MVC project:

```
dotnet new mvc
dotnet new gitignore
```

4. Then git-initate the folder and add the files to Git staging area:

```
git init
git add .
```
git commit -m “Initiated webapp project folders and .net mvc structure”

Setting up a Git Repository

If you do not already have SSH keys set up, please refer to Module X. You have the option at this point to either create a repository using the graphical interface at github.com, grab the SSH key there and do a git remote add-command to connect your folder to your repository.

In this tutorial however, just like with the MVC project, we will not use graphical interfaces (as we are aiming to automate processes in the future), here we cover how to set up a repository using Bash.

First: Check if you have GitHub CLI installed:

```
gh –version
```

If it is not installed, you will see “bash: gh: command not found”. Check Module X for installation instructions!

Now, let us proceed to initialize the repo:

```
gh repo create YourRepoName –public –source=. –remote=origin –push
```

Then just make sure you push up the initialized project. After this step I would recommend swapping to a development branch instead:

```
git push -u origin main
git checkout -b dev
```

Add your name to the landing (index) page:

This can be done in several ways, but the simplest would be:

Open Views/Home/index.cshtml and refactor the text to add your name at chosen location. The “ViewData” section is the header showing on top of the browser and the text segment below is html-code. <h1> is a header and <p> is a paragraph. See endnote example:[i]

Alterating the text like I did in the example above will result in something like this (see endnote).[ii] Another option could be to edit the html inside Views/Shared/_Layout.cshtml

Module 2: Provision a hosting environment

Like the index page name-adding, we have multiple ways to provision a hosting environment in Azure. The obvious one would be to use the Azure platform! But if we want to take this one step further, and prepare for automate deployment, it would be better to use Azure CLI.

Taking this even one step further, we might find ourselves in a situation where we would want multiple similar setups, then a re-usable template would be great. For that purpose, we have ARM templates (and to extend that, tools like for example, Bicep).

For this tutorial, I will guide you through a basic ARM template provision.

What is ARM (Azure Resource Manager)?

It is the deployment and management service for Azure. It can be accessed through Azure Portal (easy to use interface), Azure CLI (cross-platform and great for scripting and automation), PowerShell Modules (extended tooling but Windows centric), REST API (limitations to automation and complex) and the SDK (similar to API but less flexible).

ARM templates are written in JSON and are declarative, which means instead of having an ordered list of tasks for the system (imperative), we tell the system “this is what we want it to do” and the system will figure out how to get there. Since it is quite a potent tool, it can be challenging and complex to write bigger ARM templates. Thus, there are tools like Bicep. We will not expand into that or cover it in this template.

There is a section in Module X where I describe how to set up a basic ARM template. For now we will create a JSON file named *vm_arm_parameters.json* – see code here.

Even though ARM templates are idempotent – which means we can replace code and run a deploy command over and over with it just adjusting the changes (and thus would be able to deploy changes one at a time), we will use *vm_arm_template.json* – code here.

Now

Shit hits the fan: restart the RG

You can run this command to delete the resource group, in case you need to start over:

```
az group delete –resource-group TemplateRG
```

Make sure to adjust “TemplateRG” to whatever name you gave your RG.

Module 3: Configure the environment

Answer here

“Installera .NET runtime samt skapa en servicefil så att du kan starta din applikation som en service”

Module 4: Deploy the application

Answer here

“Driftsätt applikationen (deploy) i värdmiljön du skapat och konfigurerat i föregående uppgifter”

Module 5: Verify your solution

Answer here

“Verifiera att webapplikationen fungerar och att den kan köra i molnet, samt att den är nåbar från internet.

Ta en skärmdump från landningssidan med ditt namn på. Se till att adressraden i webbläsaren går att läsa tydligt. Klistra in skärmdumpen på första sidan i din inlämnade rapport.

Tänk på att hela rapporten skall vara i PDF-format innan den lämnas in.

Tips: I själva huvuddelen av rapporten behöver du bara redovisa intressanta delar av skärmdumpar eller kodavsnitt. Fullständig kod kan med fördel läggas allra sist i rapporten.”

Module X: In-depth guides

GitHub SSH key-installation instructions:

Here be dragons (TODO)

Azure SSH key-installation instructions:

Here be dragons (TODO)

GitHub CLI-installation instructions:

Install GitHub CLI using winget (Windows), homebrew (Mac) or whatever Linux distro you use. For Windows:

```
winget install –id GitHub.cli
```

One problem that could happen here, which I often see in Windows 11 Pro, is that the PATH often does not work. This means the system cannot find the file. For example, I can clearly see this: C:\Program Files\GitHub CLI\ has a gh.exe, but searching for it will return nothing!

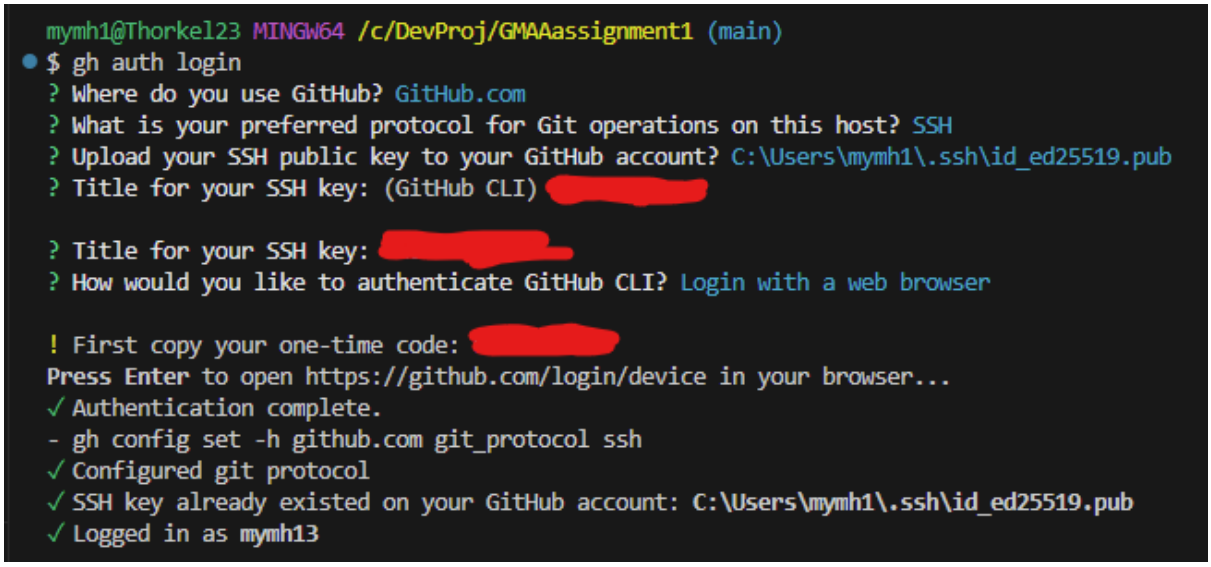
You solve this by doing the following:

1. Win + R, type sysdm.cpl and Enter (see image in endnotes).[iii]
2. Go to the Advanced tab. Click Environment Variables.
3. In the lower window (system variables), scroll down and select the Path variable.
4. Click Edit. In the new window, click New and add: C:\Program Files\GitHub CLI\
5. Click OK in all windows to close them, then close all your open terminals by typing "exit". Re-start a terminal and type "gh –version", it should be something like:

```
gh version 2.67.0 (2025-02-11)
https://github.com/cli/cli/releases/tag/v2.67.0
```

At this point you need to make sure gh is authorized to log into your GitHub account. Type the following and then watch the image below for choices:

gh auth login



Title is your choice of title for the SSH key, choice of authentication is easier by just opening a web browser and accepting GH to access your account, otherwise you have to create a token and that is slightly more complicated. Follow the instructions and you should be set.

Create a basic ARM template (ref.material see endnote):[iv]

1. Open the Azure Portal and navigate to Resource Groups.
2. Click "create a new resource group".
3. Chose a template resource group name like "TemplateRG".
4. Chose a region you are likely to use like "Sweden Central".
5. Click "review + create" but do not click Create!
6. Now click the "automation link" (see endnote picture).[v]
7. On the Template page, we have two documents:

- Template (defines the resources required to create the RG)
- Parameters (values used in the template)

8. Click the Download button above the Template tab.
9. Extract the zip where you want your project.

At this point you can set the parameter values in your project in the Parameters json-file and use the Template to define resources required for the resource group (RG).

You can use this code to deploy the jsons through the Azure CLI:

```
az deployment sub create –location swedencentral –template-file
template.json –parameters parameters.json
```

What does this do? It creates a resource group, as that is what is the template we have initiated and the command we used. To create another ARM template we will have to adjust the Template (what are we creating) and the Parameters (values for the content).

ARM parameters.json

```
{
    "$schema": "https://schema.management.azure.com/schemas/2018-05-
01/subscriptionDeploymentTemplate.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "rgName": {
            "type": "string"
        },
        "rgLocation": {
            "type": "string"
        },
        "tags": {
            "type": "object",
            "defaultValue": {}
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.Resources/resourceGroups",
            "apiVersion": "2018-05-01",
            "location": "[parameters('rgLocation')]",
            "name": "[parameters('rgName')]",
            "properties": {},
            "tags": "[parameters('tags')]"
        }
    ],
    "outputs": {}
}
```

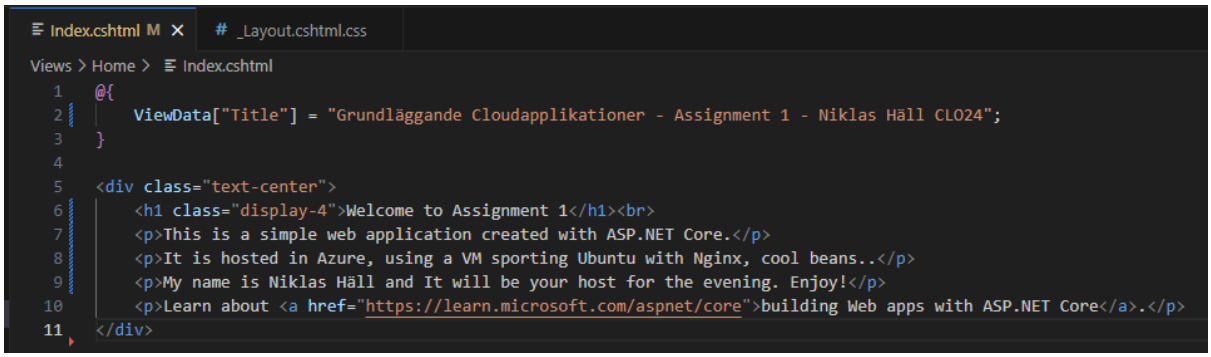
ARM template.json

Cody yodu hery

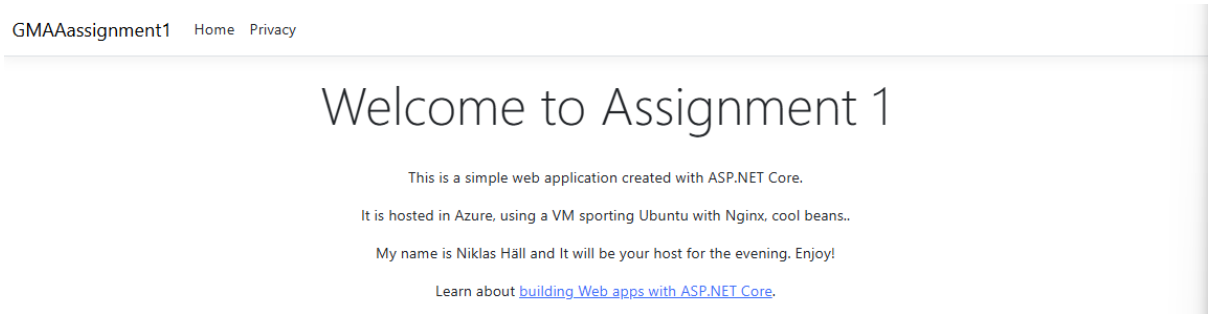
Reference / Endnotes:

---

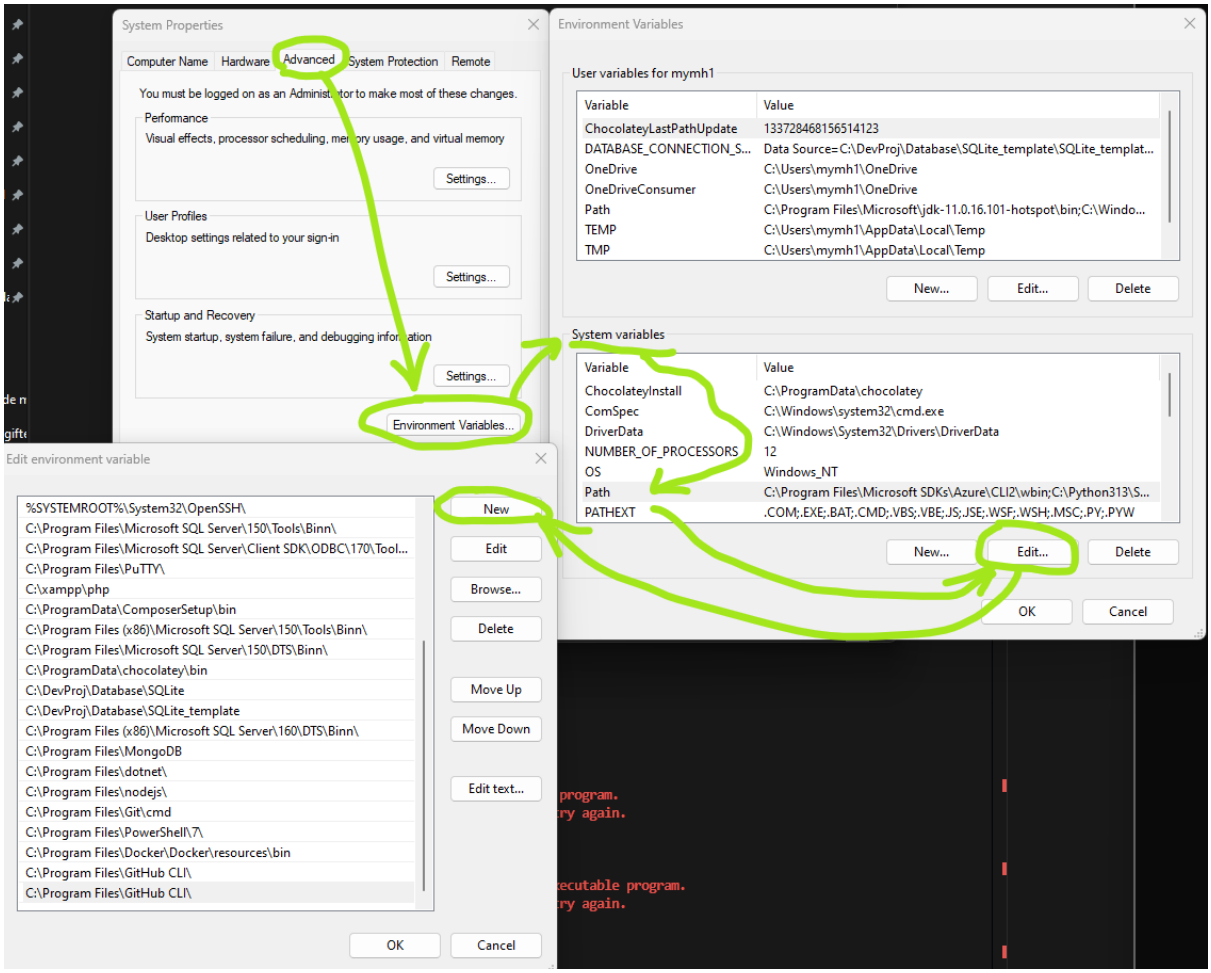
i



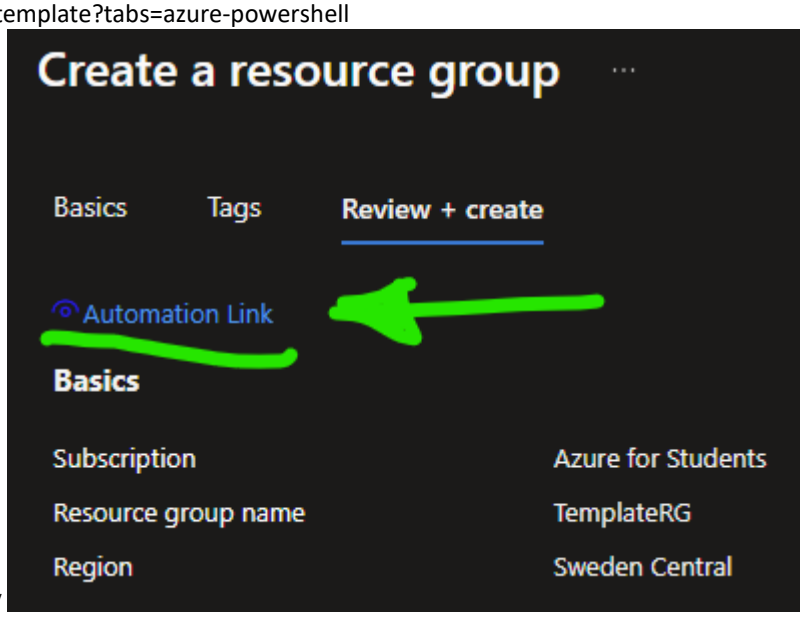
ii



iii



[iv] https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/template-tutorial-create-first-template?tabs=azure-powershell



[v]