

Cloud Project

By Roy Gebrayel

20211125

Overview

1 - created all the resources : sql database , sql server , keyvault , webapp , functionapp , docker , acr , self hosted vm , terraform , test case , storage account

2 - created the web app in golang and connected it with the function app written in nodejs

3 - connecting the web app with the sql database on the login post api (written as a trigger in the nodejs function app)

4 - connecting the web app with the storage account on the upload-file api ... the upload-file api uploads a file from your local pc to the storage account (written as a trigger in the nodejs function app)

5 - Written tasks in the azure boards and mark them as complete when i complete them

6- Written a pipeline that triggers on master the building and deployment to the acr and release of the logingo project

7 - Tried the test case

8 - tried to use terraform to understand it and managed it correctly

1 - RESOURCES CREATION

1 - vm creation :

The screenshot shows the Microsoft Azure Virtual Machines dashboard. On the left, there's a sidebar with navigation links like Home, Create, and Settings. The main area displays a single virtual machine named "roy-vm". The machine details are as follows:

- Virtual machine**: Computer name is "roy-vm", operating system is "Linux (ubuntu 20.04)", image publisher is "canonical", image offer is "0001-com-ubuntu-server-focal", and image plan is "20_04-lts-gen2".
- Networking**: Public IP address is "172.203.194.1" (Network interface "roy-vm658_z1"), private IP address is "-".
- Size**: Standard B2ms, 2 vCPUs, 8 GiB RAM.
- Disk**: OS disk "roy-vm_OsDisk_1_d92336e49a5447f9893606bb65762b0f".

Created the vm using ubuntu 20.04 linux os and configured to enable docker and install it

The screenshot shows the Azure DevOps Agent pools settings page. On the left, there's a sidebar with Organization Settings and various project management links. The main area shows existing agent pools "Azure Pipelines" and "Default". A modal window titled "Add agent pool" is open, prompting for a pool type (set to "Self-hosted"), a name ("az400m05i05a-pool"), and a description. Pipeline permissions are set to "Auto-provision this agent pool in all projects".

Add agent pool to the logingo

Connecting the self agent server to the logingo web app

2 - function app :

```
Q roy@archlinux:~/express-backend-function
Deployment completed successfully.
[2023-12-04T09:40:28.523Z] Syncing triggers...
Functions in functionnodeapp:
  CredentialsPosting - [httpTrigger]
    Invoke url: https://functionnodeapp.azurewebsites.net/api/credentialsposting

  GetRequest - [httpTrigger]
    Invoke url: https://functionnodeapp.azurewebsites.net/api/users

[roy@archlinux express-backend-function]$ code .
[roy@archlinux express-backend-function]$ code .

[roy@archlinux express-backend-function]$ 
[roy@archlinux express-backend-function]$ func new --template "Timer Trigger" --name TimeTrigger
Select a number for template:Timer Trigger
Function name: [TimeTrigger] Writing /home/roy/express-backend-function/TimeTrigger/index.js
Writing /home/roy/express-backend-function/TimeTrigger/readme.md
Writing /home/roy/express-backend-function/TimeTrigger/function.json
The function "TimeTrigger" was created successfully from the "Timer Trigger" template.
Did you know? The new Node.js programming model is generally available. Learn how you can try it out today at https://aka.ms/AzFuncNodeV4
[roy@archlinux express-backend-function]$ func azure functionapp publish functionnodeapp
Getting site publishing info...
[2023-12-04T11:20:27.924Z] Starting the function app deployment...
Creating archive for current directory...
Uploading 9.05 MB [#####
Upload completed successfully.
Deployment completed successfully.
[2023-12-04T11:21:12.403Z] Syncing triggers...
Functions in functionnodeapp:
  CredentialsPosting - [httpTrigger]
    Invoke url: https://functionnodeapp.azurewebsites.net/api/credentialsposting

  GetRequest - [httpTrigger]
    Invoke url: https://functionnodeapp.azurewebsites.net/api/users

  TimeTrigger - [timerTrigger]

[roy@archlinux express-backend-function]$ 
```

Created four triggers with the func in the terminal

CredentialsPosting.js - express-backend-function - Visual Studio Code

```

File Edit Selection View Go Run Terminal Help
EXPLORE index.js function.json CredentialsPosting function.json GetRequest CredentialsPosting.js
EXPRESS-B... .vscode CredentialsPosting > <unknown> > exports
Click here to ask Blackbox to help you code faster!
const isValidLogin = (password) => {
    // Ensure the password is at least 8 characters long
    return password.length >= 8;
};

Code Suggestions
Comment Code
module.exports = async function (context, req) {
    context.log("CredentialsPosting function processed a request.");

    if (req.method === "POST") {
        const { username, password } = req.body;

        // Validate login credentials
        if (!isValidLogin(password)) {
            // Render a failed login response
            context.res = {
                status: 400,
                body: "Invalid credentials",
            };
        } else {
            // Handle successful login
            context.res = {
                status: 200,
                body: `Welcome, ${username}!`,
            };
        }
    } else {
        // Handle other requests
        context.res = {
            status: 404,
            body: "Method not supported",
        };
    }
};

```

functionnodeapp - Microsoft Edge YouTube

portal.azure.com/?Microsoft_Azure_Education_correlationId=9fe8c4fb-13f0-46f3-8249-6438b3e13771&Microsoft_Azure_Education...

Microsoft Azure Search resources, services, and docs (G+)

Home >

functionnodeapp Function App

Search Overview Activity log Access control (IAM) Tags Diagnose and solve problems Microsoft Defender for Cloud Events (preview)

Subscription (move) : Roy Gebrayel Git Username :
Subscription ID : 189af0e7-fa98-4751-b5a8-0d39f4d49d5f Git clone url : https://functionnodeapp.scm.azurewebsites.net.git
Runtime version : 4.27.5.21549

Tags (edit) :

Functions Metrics Properties Notifications (0)

+ Create Set up local environment Refresh

Filter by name...

Name	Trigger	Status	Monitor
CredentialsPosting	HTTP	Enabled	Invocations and more
GetRequest	HTTP	Enabled	Invocations and more
queue1	Queue	Enabled	Invocations and more
Timer	Timer	Enabled	Invocations and more

Created the function app in vs code with node js runtime stack and created four triggers
 CredentialsPosting (post http request) , GetRequest (get http request) , Timer (timer trigger) and queue1 (azure queue storage trigger)

```

roy@archlinux:~/go-test/functionnodeapp
return self.handler(*args, **kwargs)
File "/opt/azure-cli/lib/python3.11/site-packages/azure/cli/core/commands/command_operation.py", line 112, in handler
    client = self._client_factory(self._cli_ctx, command_args) if self._client_factory else None
File "/opt/azure-cli/lib/python3.11/site-packages/azure/cli/command_modules/storage/_client_factory.py", line 348, in cf_queue_client
    return cf_queue_service(client._ctx, kwargs).get_queue_client(queue=kwarg.pop('queue_name'))
File "/opt/azure-cli/lib/python3.11/site-packages/azure/cli/command_modules/storage/_client_factory.py", line 339, in cf_queue_service
    return t_queue_service.from_connection_string(conn_str=connection_string, **client_kwargs)
File "/opt/azure-cli/lib/python3.11/site-packages/azure/multiapi/storagev2/queue/v2018_03_28/_queue_service_client.py", line 148, in from_connection_string
    account_url, secondary, credential = parse_connection_str(
                                                ^^^^^^^^^^
File "/opt/azure-cli/lib/python3.11/site-packages/azure/multiapi/storagev2/queue/v2018_03_28/shared/base_client.py", line 401, in parse_connection_str
    raise ValueError("Connection string missing required connection details.")
ValueError: Connection string missing required connection details.
To check existing issues, please visit: https://github.com/Azure/azure-cli/issues
[roy@archlinux functionnodeapp]$ az storage message put --queue-name queue1 --content "Hello, Azure Queue!" --connection-string "DefaultEndpointsProtocol=https;AccountName=functionnodeappd7e6;AccountKey=j/3Lb3aaOTFWSSyHuPg+Gu9Lk0uzwjCRFOXVkpfo/av00VutHceLR647tA3rZqoItd56s330BFE+AStAbTvFw==;EndpointSuffix=core.windows.net"
Command group 'storage message' is in preview and under development. Reference and support levels: https://aka.ms/CLI_refstatus
The specified queue does not exist.
RequestId:4616e361-7003-0036-6bed-283277000000
Time:2023-12-07T09:15:01.3889386Z
ErrorCode:QueueNotFound
[roy@archlinux functionnodeapp]$ az storage message put --queue-name queue --content "Hello, Azure Queue!" --connection-string "DefaultEndpointsProtocol=https;AccountName=functionnodeappd7e6;AccountKey=j/3Lb3aaOTFWSSyHuPg+Gu9Lk0uzwjCRFOXVkpfo/av00VutHceLR647tA3rZqoItd56s330BFE+AStAbTvFw==;EndpointSuffix=core.windows.net"
Command group 'storage message' is in preview and under development. Reference and support levels: https://aka.ms/CLI_refstatus
{
    "content": "Hello, Azure Queue!",
    "dequeueCount": null,
    "expirationTime": "2023-12-14T09:15:21+00:00",
    "id": "b50f9d93-054d-4835-9761-eabe1fb73dda",
    "insertionTime": "2023-12-07T09:15:21+00:00",
    "popReceipt": "AgAAAAAAAAMAAAAAAA22ob5+0o2gE=",
    "timeNextVisible": "2023-12-07T09:15:21+00:00"
}
[roy@archlinux functionnodeapp]$ 

```

Sending message to the queue

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is `portal.azure.com/?Microsoft_Azure_Education.correlationId=9fe8c4fb-13f0-46f3-8249-6438b3e13771&Microsoft_Azure_Educatio...`. The page title is "queue | Monitor". On the left, there's a sidebar with "Developer" tools: "Code + Test", "Integration", "Monitor" (which is currently selected), and "Function Keys". The main content area has tabs for "Invocations" and "Logs". The "Logs" tab is active, showing a single log entry: "Connected!". Above the logs, there are buttons for "App Insights Logs", "Log Level", "Stop", "Copy", "Clear", "Open in Live Metrics", and "Leave Feedback".

Here we see that the queue is connected

3 - webapp :

The screenshot shows the Microsoft Azure portal interface for the 'App Services' section. The main page displays the 'logingo' web app, which is a 'Web App'. The left sidebar lists other services like 'functionnodeapp' and 'logingo'. The right panel provides detailed information about the app, including its resource group ('roygebrayel-rg'), status ('Running'), and location ('East US'). It also shows the 'Default domain' as 'logingo.azurewebsites.net' and the 'App Service Plan' as 'ASP-roygebrayelRG-9a5b (B1: 1)'. The 'Properties' tab is selected, showing the 'Web app' settings with the name 'logingo'. A 'JSON View' button is available in the top right corner.

Created the logingo web app and written it in golang also built it with docker and pull it from the acr

4 - SQL database :

The screenshot shows the Microsoft Azure portal interface for the 'SQL databases' section. The main page displays the 'roysqlDb' database, which is a 'SQL database'. The left sidebar lists other databases like 'functionnodeapp' and 'roysqlDb'. The right panel provides detailed information about the database, including its resource group ('roygebrayel-RG'), status ('Online'), and location ('East US'). It also shows the 'Server name' as 'sqlservergo.database.windows.net' and the 'Elastic pool' as 'No elastic pool'. The 'Properties' tab is selected, showing the 'Getting started' section with a call to action: 'Start working with your database'. A 'JSON View' button is available in the top right corner.

Created a sql db with the sql authentication password

5 - Storage account :

The screenshot shows the Microsoft Azure Storage accounts page. On the left, there's a list of storage accounts: 'functiongorogyo600087', 'functionnodeappa1d7e6', 'goaccountstorage', 'goroystorage' (which is selected), and 'roygebrayelrgbfae'. The main pane displays the details for 'goroystorage', including its resource group ('roygebrayel-RG'), location ('eastus'), subscription ('Roy Gebrayel'), and disk state ('Available'). It also shows performance settings ('Standard'), replication ('Locally-redundant storage (LRS)'), account kind ('StorageV2 (general purpose v2)'), and provisioning state ('Succeeded'). A 'Tags' section is present, and a 'Properties' tab is selected. At the bottom, there are links for 'Blob service' and 'Security'.

Created the storage account named go roy storage

The screenshot shows the Microsoft Azure Storage containers page for the 'uploadedfiles' container. The left sidebar includes options like 'Overview', 'Diagnose and solve problems', 'Access Control (IAM)', 'Settings', 'Shared access tokens', 'Access policy', 'Properties', and 'Metadata'. The main pane shows a list of blobs with columns: Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. The blobs listed are all named 'Screenshot from 202...' and have sizes ranging from 38.93 KiB to 156.04 KiB.

This is the uploaded files from the web to the storage account

6 - acr

The screenshot shows the Microsoft Azure Container Registry interface. On the left, a sidebar lists 'Container registries' and the selected 'goprojects' registry. The main pane displays the 'Overview' tab for the 'goprojects' registry. Key details shown include:

- Resource group: [roygebrayel-RG](#)
- Location: East US
- Subscription: [Roy Gebrayel](#)
- Subscription ID: 189af0e7-fa98-4751-b5a8-0d39f4d49d5f
- Provisioning state: Succeeded
- Pricing plan: Standard

The 'Get started' tab is selected. A banner at the bottom right reads: "Simplify container lifecycle management" and "Container registry allows you to build, store, and manage container images and".

Deployed the docker container into the registry

7 - keyVault :

The screenshot shows the Microsoft Azure Key Vault interface. On the left, a sidebar lists 'Key vaults' and the selected 'keyvaultroy' vault. The main pane displays the 'Overview' tab for the 'keyvaultroy' vault. Key details shown include:

- Resource group: [roygebrayel-RG](#)
- Location: East US
- Subscription: [Roy Gebrayel](#)
- Subscription ID: 189af0e7-fa98-4751-b5a8-0d39f4d49d5f
- Vault URI: <https://keyvaultroy.vault.azure.net/>
- Sku (Pricing tier): Standard
- Directory ID: e731378a-11b1-405c-bb1c-c047ae5a5d54
- Directory Name: Université Antonine (UA)
- Soft-delete: Enabled
- Purge protection: Disabled

The 'Get started' tab is selected.

Created a keyvault and inserted secrets

2 - Web App

1 - Content of the web app :

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "GO-TEST". Files include: .terraform, .vscode, build, connections, database, storage, templates, .gitignore, .terraform.lock.hcl, Dockerfile, go.mod, go.sum, login, main.go, main.test.go, main.tf, main.tf.txt, README.md, and resources.sh.
- Code Editor:** The main editor window displays the "main.go" file. The code is written in Go and defines two functions: `insertUser` and `GetUsersHandler`. The `GetUsersHandler` function makes an HTTP request to an external API to fetch user data.
- Terminal:** At the bottom, the terminal shows the command "master*".
- Bottom Bar:** Includes icons for Go Update Available, Prettier, and other VS Code features.

1 - this project is written in go :

Build folder : a folder to put all the yaml pipelines in the project

Connection folder : a folder having all the sql db connection strings to be able to connect it with the main.go file

Templates folder : a folder having the html templates for this project

Dockerfile : a file to write my docker configuration

Main.tf : a terraform file to try it and test how it is done

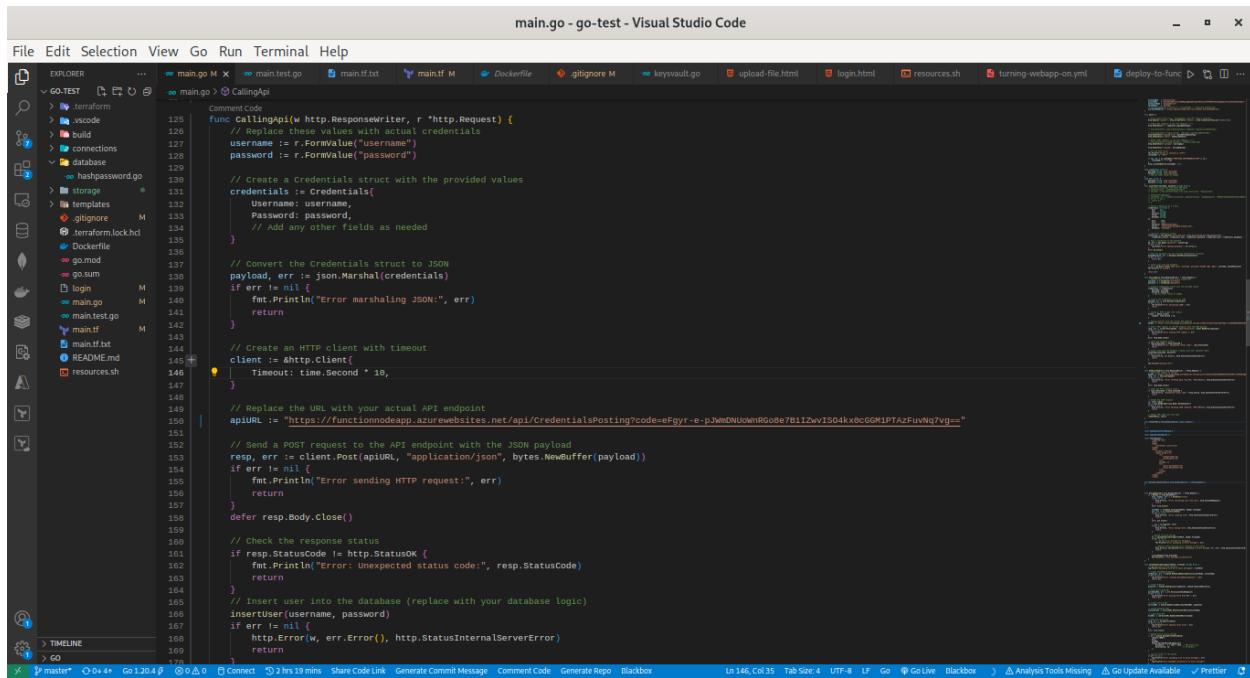
Storage folder : a folder having all the storage account logic for the go project

Main.go : the main file where i'm having all my api trigger function (getted from the function app)

2 - Azure repo link to check the project :

To check my repo please click on this link for azure devops : [logingo](#)

3 - Login api to the database



The screenshot shows the Visual Studio Code interface with the main.go file open. The code implements a login API endpoint. It starts by defining a Credentials struct and marshaling it to JSON. It then creates an HTTP client with a timeout and sends a POST request to an API endpoint. Finally, it inserts the user credentials into a database. The code includes comments explaining each step.

```
main.go - go-test - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER ... main.go M main-test.go main.tf.txt main.tf M Dockerfile gitignore M upload-file.html login.html resources.sh turning-webapp-on.yml deploy-to-func ...
Comment Code
func CallingApi(w http.ResponseWriter, r *http.Request) {
    // Replace these values with actual credentials
    username := r.FormValue("username")
    password := r.FormValue("password")

    // Create a Credentials struct with the provided values
    credentials := Credentials{
        Username: username,
        Password: password,
        // Add any other fields as needed
    }

    // Convert the Credentials struct to JSON
    payload, err := json.Marshal(credentials)
    if err != nil {
        fmt.Println("Error marshaling JSON:", err)
        return
    }

    // Create an HTTP client with timeout
    client := &http.Client{
        Timeout: time.Second * 10,
    }

    // Replace the URL with your actual API endpoint
    apiURL := "https://functionnodeapp.azurewebsites.net/api/CredentialsPosting?code=eFgyr-e-pJmDNlwhRGo7B1IZwvIS04kx0cGGM1PTAzFuvNq7vg=="

    // Send a POST request to the API endpoint with the JSON payload
    resp, err := client.Post(apiURL, "application/json", bytes.NewBuffer(payload))
    if err != nil {
        fmt.Println("Error sending HTTP request:", err)
        return
    }
    defer resp.Body.Close()

    // Check the response status
    if resp.StatusCode != http.StatusOK {
        fmt.Println("Error: Unexpected status code:", resp.StatusCode)
        return
    }
    // Insert user into the database (replace with your database logic)
    insertUser(username, password)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
}
```

This function calls the api trigger from the function app (written in the apiURL) , this is a post request that when a user write its credentials in the login form it call the insertuser function and send it to the sql db

```
functionHandler = async (req) => {
    const sql = require("mssql");
    const config = {
        user: "roy",
        password: "R0gez1313131313",
        server: "sqlservergo.database.windows.net",
        database: "roysqldb",
        options: {
            encrypt: true, // Use this if you're connecting to Azure SQL Database
        },
    };

    try {
        // Create a connection pool
        const pool = await sql.connect(config);

        // Execute the query
        const result = await pool
            .request()
            .query("SELECT username, password FROM users");

        // Map the result to an array of users
        const users = result.recordset.map((row) => ({
            username: row.username,
            password: row.password,
        }));

        // Respond with the list of users
        context.res = {
            status: 200,
            body: JSON.stringify(users),
        };
    } catch (error) {
        // Handle errors
        context.res = {
            status: 500,
            body: JSON.stringify({ error: error.message }),
        };
    } finally {
        // Close the connection pool
        sql.close();
    }
};
```

This function in js connects into the db and query it to select all users in the table users

```
func insertUser(username, password string) error {
    // Define credentials as a struct
    credentials := struct {
        port     int
        user     string
        password string
        server   string
        database string
    }{
        port:     1433,
        user:    "roy",
        password: "R0gez1313131313",
        server:  "sqlservergo.database.windows.net",
        database: "roysqldb",
    }

    // Construct connection string
    connString := fmt.Sprintf("server=%s;user id=%s;password=%s;port=%d;database=%s",
        credentials.server, credentials.user, credentials.password, credentials.port, credentials.database)

    // Open a connection to the database
    db, err := sql.Open("sqlserver", connString)
    if err != nil {
        log.Fatal("Error opening database:", err.Error())
    }
    defer db.Close()

    // Hash the password (use the existing hashPassword function)
    hashedPassword, err := database.hashPassword(password)
    if err != nil {
        return err
    }

    // Insert user into the database
    _, err = db.Exec("INSERT INTO users (username, password) VALUES (?, ?)", username, hashedPassword)
    if err != nil {
        fmt.Println("It worked")
    }

    return err
}
```

This is the insertUser function

```

index.js - express-backend-function - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPRESS.js ... index.js x ⌂ function.json CredentialsPosting.js ⌂ function.json GetRequest ⌂ CredentialsPosting.js
GetRequest > Click here to ask BlaBlaCloud to help you code faster!
1 const sql = require("ms-sql");
2
3 const config = {
4   user: "Rogege31313131313",
5   password: "Rogege31313131313",
6   server: "sqlservergo.database.windows.net",
7   database: "roysqldb",
8   options: {
9     encrypt: true, // Use this if you're connecting to Azure SQL Database
10   },
11 };
12
13 Comment Code
14 module.exports = async function (context, req) {
15   try {
16     // Create a connection pool
17     const pool = await sql.connect(config);
18
19     // Execute the query
20     const result = await pool
21       .request()
22       .query("SELECT username, password FROM users");
23
24     // Map the result to an array of users
25     const users = result.recordset.map((row) => ({
26       username: row.username,
27       password: row.password,
28     }));
29
30     // Respond with the list of users
31     context.res = {
32       status: 200,
33       body: JSON.stringify(users),
34     };
35   } catch (error) {
36     // Handle errors
37     context.res = {
38       status: 500,
39       body: JSON.stringify({ error: error.message }),
40     };
41   } finally {
42     // Close the connection pool
43     sql.close();
44   }
45 };

```

File Explorer: EXPRESS.js, index.js, function.json, CredentialsPosting.js, GetRequest, CredentialsPosting.js, node_modules, src functions, TimeTrigger, Juncignore, package.json, host.json, local.settings.json, package-lock.json, package.json.

Timeline: 2 hrs 19 mins, Share code link, Generate Commit Message, Comment Code, Generate Repo.

Bottom bar: Line numbers, CodeLens, Spaces:2, UTF-8, CR/LF, JavaScript, Go Live, Blackbox, prettier.

Also another function in the function app to fetch the users and **ENCRYPTED** passwords on the webapp on the "/users" path

The flow of the database query in the logingo web app :

localhost | logingo | Login Page | logingo | Projects | Azure/az | Project c | Develop | (4) What | + | Relaunch to update

logingo.azurewebsites.net

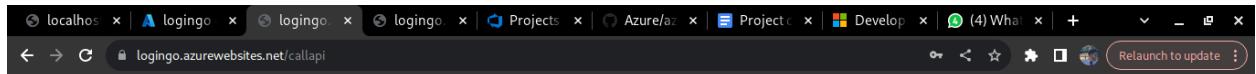
Login

Username:

Password:

Login

The user enters the username and password in the login form



When clicked on the login .. the loggingo webapp calls the post request on the /callapi path
It saves the users credentials to the sql database in azure

A screenshot of the Microsoft Azure portal. The URL in the address bar is 'portal.azure.com/#@ua.edu.lb/resource/subscriptions/189af0e7-fa98-4751-b5a8-0d39f4d49d5f/resourceGroups/roygebrayel-RG...'. The main content is the 'roysqlldb (sqlservergo/roysqlldb)' database page in the Azure Data Studio Query editor (preview). The left sidebar shows database management options like 'Compute + storage', 'Connection strings', and 'Properties'. The right pane shows the 'Tables' section with 'dbo.users' selected. A query is running in the editor: 'SELECT TOP (1000) * FROM [dbo].[users]'. The results show four rows of data:

21	roy.gebrayel11@gm...	\$2a\$10\$zQlYKefM	
22	marita	\$2a\$10\$M6/08KujE	
23	roygebrayel	\$2a\$10\$MHL63G2k	
24	roy.gebrayel11@gm...	\$2a\$10\$O9gGYh	

At the bottom of the results pane, it says 'Query succeeded | 0s'.

This is the database queried to show all users from the users table

User List

Username	Password
d;km;zmclzmc;l	\$2a\$10\$8pJBlz6m.zLAaybY3VY4m.pHGF14yAX04CPzXZk8wypcsjoc7COy
roygebrayel	\$2a\$10\$lemCFiAMEIK0c9xs4tyVp0LznZbo0.GH6ZlRcZnSPstfxmsij/W
lsknvlkznvlkdvn	\$2a\$10\$1aJHWvRtDc9R60kQvxZeKCBl3uvKXXYRp670UbjhH15dfDiG
lsknvlkznvlkdvn	\$2a\$10\$x9YCbVLFLkavSpS5.WljievlCpbw2xErWo/Cmo1vg655cd9iKV5E2
lsknvlkznvlkdvn	\$2a\$10\$sdICvR1vcCggbZS.G0bRoX07GJuUmarbfcaDq1zaFplvazEAfulwW
lsknvlkznvlkdvn	\$2a\$10\$FzMejhOostkD8UwhTOEPNultiR0u6e04txTlhG7Cbo7gW0Z/IS
lsknvlkznvlkdvn	\$2a\$10\$F50t0cr0M4MybDeYGfkc..7y.RPDjcPoNuUZelQZsaDm4jW6hGP2
lsknvlkznvlkdvn	\$2a\$10\$SOledYuQULE.n0W68XX.QOsld8P5RyhsgokIr6xpouD9auec7qa
roy.gebrayel11@gmail.com	\$2a\$10\$HOO2lrmPbHgfCd4GMuCj.YO159mfXvWgle3Vm1DtrBrFyX2b1p9m
roy.gebrayel11@gmail.com	\$2a\$10\$ss3t5yvDSSy0dq1TVVikOpT5YnmkHIRfcfmZEKQXoVVsc3Cdkd2
sdhnlndckak.cn	\$2a\$10\$AcXs.cvfIEe3A9TeUj.OHQvm1hC4uF02PuthOsLYKCrzDou2K2a
lzcnczncknz	\$2a\$10\$faobB7vrmrPOX6kBSSsq0WdfUIKscck1z9AYpxZkpH3U6/8W7XG
roy.gebrayel11@gmail.com	\$2a\$10\$HVIEWSAqylnK.Hacs1XdyVut1PabnINlqixG/SMTaqv4/5JASocUDG
roy.gebrayel11@gmail.com	\$2a\$10\$SMYm74ammBzu.w9PjiaoG6YEpMCHu5Qqme3k7bdIvSPkp7q.6Kqxe
roy.gebrayel11@gmail.com	\$2a\$10\$RaMoIMw1KZ7Ln7LOAwV2BOMuJklisQw4XrHvOMEYzpqFWqWSfqzw2
roy.gebrayel11@gmail.com	\$2a\$10\$SSUbjnCsw/Wgn7wLUIm9n/elqTIdckDDInU4B1N8Qb.tjVeIRfQx1G
gisele	\$2a\$10\$pi4MrQwUQnKZ1XXzgy5KdubwYi7weGmmmwUE.YYukhAPGweBIXXeS
andrew.gebrayel	\$2a\$10\$1XAxzsl2GifTzID/AT017uRxasUmcYFPIm08Pvv836XVY5u.E4BC
antiro313	\$2a\$10\$g2dl11cgpyhunwU0/HW5Setzn2YGjNhbxDF4MQuxxNi6S31dSgkhSe
roy.gebrayel11@gmail.com	\$2a\$10\$MbXQbrcYMbu1kwExnUJA0oo0.dKPJ.qjoQTkhJ06HGPFm/g9v/5F6
roy.gebrayel11@gmail.com	\$2a\$10\$SzQIYK_efM6WEwkpYLLDo.g/NMdyw2GsdmXKnUC3ysMcjG5Ca
marita	\$2a\$10\$M6/08Kj8uZucf7lkQOrupQFkPWjas701xOkEnlpa6OqechnUpfK
roygebrayel	\$2a\$10\$MH63G2kXoroCQ6.6E7U7ZoOuk7d/F0aCk0W9BN16y13edRwRGY4Wq
roy.gebrayel11@gmail.com	\$2a\$10\$v09gGYVhSb2/DnzDuFfNO4EiTThb/coXvEffHrtlxkoMw4Q2TBuvC

Here is the /users path api that fetches the users and **ENCRYPTED** passwords into the web app

4 - Storage account api in the web app

main.go - go-test - Visual Studio Code

```

File Edit Selection View Go Run Terminal Help
EXPLORE main.go M ... main.test.go main.tf.txt .gitignore M upload-file.html login.html resources.sh turning-webapp-on.yml deploy-to-func ...
main.go > InsertUser
Code Suggestions
Comment Code
func StorageUpload(w http.ResponseWriter, r *http.Request) {
    if r.Method == http.MethodPost {
        file, header, err := r.FormFile("file")
        if err != nil {
            http.Error(w, "Error retrieving file from form", http.StatusBadRequest)
            return
        }
        defer file.Close()

        filePath := filepath.Join(uploadPath, header.Filename)
        out, err := os.Create(filePath)
        if err != nil {
            http.Error(w, "Error creating file", http.StatusInternalServerError)
            return
        }
        defer out.Close()

        if err = io.Copy(out, file);
        if err != nil {
            http.Error(w, "Error saving file", http.StatusInternalServerError)
            return
        }

        // Upload to Azure Storage
        err = uploadToAzureStorage(filePath, header.Filename)
        if err != nil {
            // Log the error message for debugging
            fmt.Println("Error uploading to Azure Storage:", err)

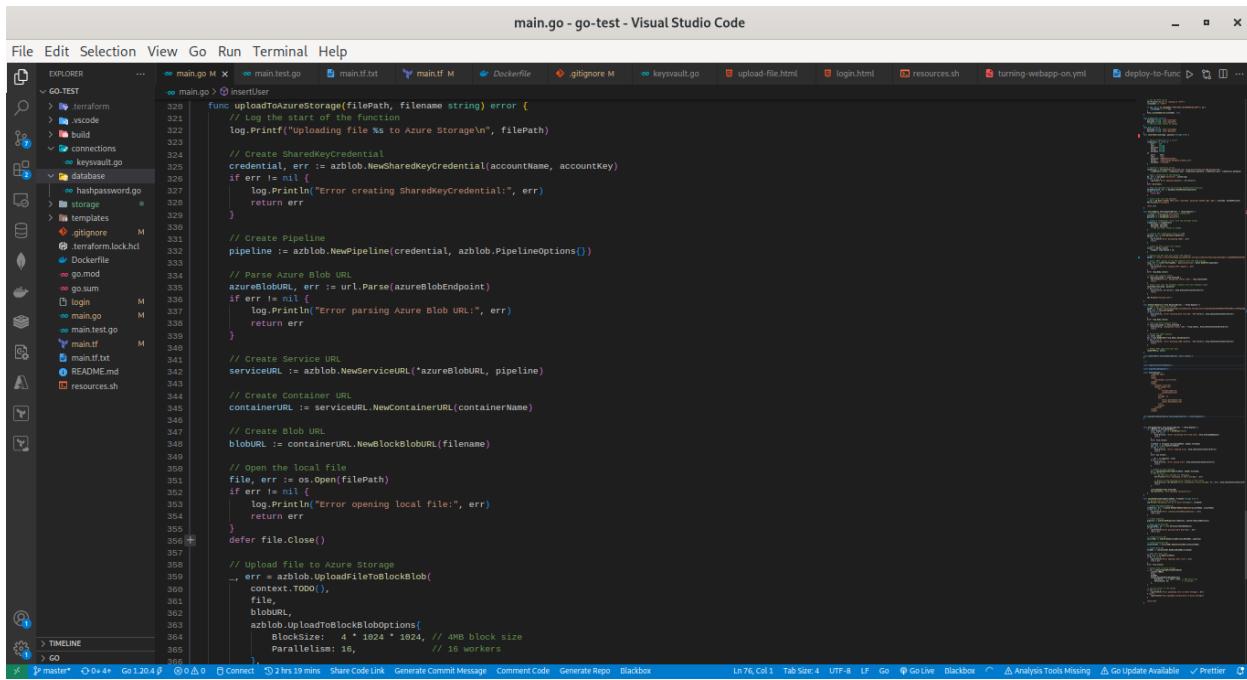
            // Return a more detailed error response to the client
            http.Error(w, fmt.Sprintf("Error uploading to Azure Storage: %v", err), http.StatusInternalServerError)
            return
        }

        w.WriteHeader(http.StatusOK)
        fmt.Printf(w, "File uploaded successfully")
    }
}

func uploadToAzureStorage(filePath, filename string) error {
    Comment Code
}

```

This function here takes the file inserted and put it inside the storage folder



The screenshot shows the Visual Studio Code interface with the main.go file open. The code is a Go function named uploadToAzureStorage that takes a filePath and filename string. It starts by logging the start of the function. It then creates a SharedKeyCredential using accountName and accountKey. If the credential creation fails, it prints an error message and returns. Next, it creates a Pipeline using the credential and azblob.PipelineOptions. It then parses the Azure Blob URL from the filePath. If parsing fails, it prints an error message and returns. It creates a Service URL using azblob.NewServiceURL with the parsed blob endpoint and pipeline. It also creates a Container URL using the service URL and container name. Then, it creates a Blob URL using the container URL and the provided filename. It opens the local file using os.Open and checks if it fails. If it does, it prints an error message and returns. Finally, it uploads the file to Azure Storage using azblob.UploadFileToBlockBlob with context.TODO(), the file, the blob URL, and options for block size (4 * 1024 * 1024) and parallelism (16).

```
func uploadToAzureStorage(filePath, filename string) error {
    // Log the start of the function
    log.Printf("Uploading file %s to Azure Storage\n", filePath)

    // Create SharedKeyCredential
    credential, err := azblob.NewSharedKeyCredential(accountName, accountKey)
    if err != nil {
        log.Println("Error creating SharedKeyCredential:", err)
        return err
    }

    // Create Pipeline
    pipeline := azblob.NewPipeline(credential, azblob.PipelineOptions{})

    // Parse Azure Blob URL
    azureblobURL, err := url.Parse(azureBlobEndpoint)
    if err != nil {
        log.Println("Error parsing Azure Blob URL:", err)
        return err
    }

    // Create Service URL
    serviceURL := azblob.NewServiceURL(*azureBlobURL, pipeline)

    // Create Container URL
    containerURL := serviceURL.NewContainerURL(containerName)

    // Create Blob URL
    blobURL := containerURL.NewBlockBlobURL(filename)

    // Open the local file
    file, err := os.Open(filePath)
    if err != nil {
        log.Println("Error opening local file:", err)
        return err
    }
    defer file.Close()

    // Upload file to Azure Storage
    _, err = azblob.UploadFileToBlockBlob(
        context.TODO(),
        file,
        blobURL,
        azblob.UploadFileToBlockBlobOptions{
            Blocksize: 4 * 1024 * 1024, // 4MB block size
            Parallelism: 16,           // 16 workers
        },
    )
}
```

This function here uploaded the storage file into the storage account blob

The flow of the storage account in the web app :



File Upload

In this web page and in the /upload-file request path you can choose the file and upload data into the storage account container



In here and after you click on the upload file .. the post request is called on the /upload path and sent to the storage account container

A screenshot of the Microsoft Azure Storage Explorer interface. The left sidebar shows a tree view with "uploadedfiles" selected under "Containers". The main pane displays a list of blobs in the "uploadedfiles" container. The table has columns: Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. The "Name" column lists several "Screenshot from 202..." files. The "Modified" column shows dates ranging from December 2, 2023, to December 4, 2023. The "Size" column shows file sizes like 108.93 KiB, 114.52 KiB, etc. The "Blob type" column indicates they are all "Block blob". The "Lease state" column shows "Available" for all files.

Here is the data sent from the api into the storage account container

5 - Azure boards

The screenshot shows the Azure DevOps Boards interface for the 'login Team' project. The left sidebar includes options like Overview, Boards, Work items, Backlogs, Sprints, Queries, Delivery Plans, Analytics views, Repos, Pipelines, Test Plans, and Project settings. The main area displays a Kanban board with three columns: To Do, Doing, and Done. The 'To Do' column contains five tasks: 813 (having an API that inserts username and password to the SQL database), 812 (using Jenkins), 809 (connecting the SQL server to the web app), 807 (writing all credentials into the key vault), and 806 (finishing the function app and adding two HTTP triggers functions). The 'Doing' column contains two tasks: 811 (writing a test case) and 812 (using Jenkins). The 'Done' column contains one task: 810 (making the VM a self-host agent to the web app).

Here is the azure boards where i put all my tasks and with the tasks each test case i should have

6 - pipeline

#20231204.1 • Set up CI with Azure Pipelines
build and deploy docker

This run is retained as one of 3 recent runs by master (Branch).

Summary **Code Coverage**

Manually run by roy gebrayel

Repository and version

- login
- master f2ae89d5

Time started and elapsed

- Today at 2:45 PM
- 5m 6s

Related

- 0 work items
- 1 published

Tests and coverage

Get started

Stages **Jobs**

- Build and push stage: 1 job completed, 2m 23s
- Deploy to Azure Web...: 1 job completed, 1m 55s
- Release: 1 job completed, 27s

1 artifact

This is the pipeline result where it builds and pushes the docker and then it deploys it to azure web app and after that it releases an artifact (it automatically triggers the pipeline whenever i push the code into master branch)

```

trigger:
- master

pr:
- master

resources:
- repo: self

variables:
- Container registry service connection established during pipeline creation
- dockerRegistryServiceConnection: 'f0721b2f-90c1-4a6e-aef5-0fea091f76'
- imageRepository: 'login'
- containerRegistry: 'goprojects.azurecr.io'
- dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
- tag: '$(Build.BuildId)'
- webAppName: 'your-web-app-name' # replace with your Azure Web App name
- slotName: 'production' # replace with the deployment slot name if applicable
- vmImageName: 'ubuntu-latest'

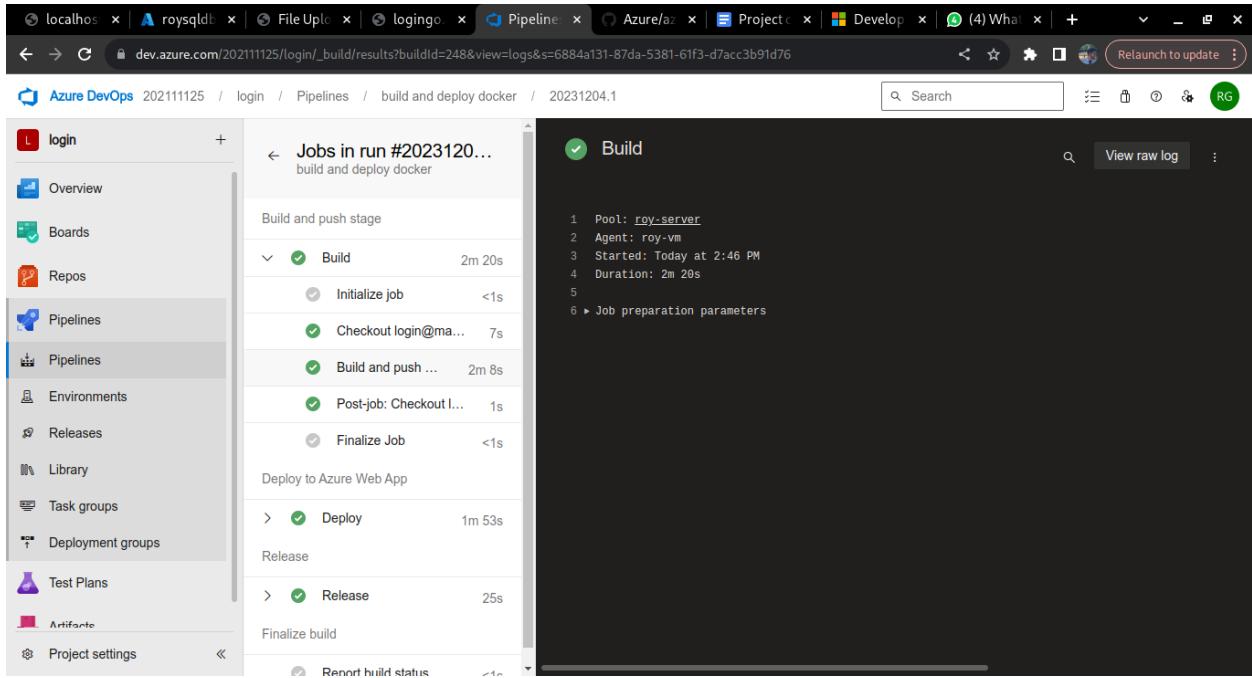
stages:
- stage: Build
  displayName: Build and push stage

```

Tasks

- .NET Core
- Android signing
- Ant
- App Center distribute
- App Center test
- Archive files
- ARM template deployment

The triggers here means that whenever i git push the code into master branch it automatically triggers the pipeline



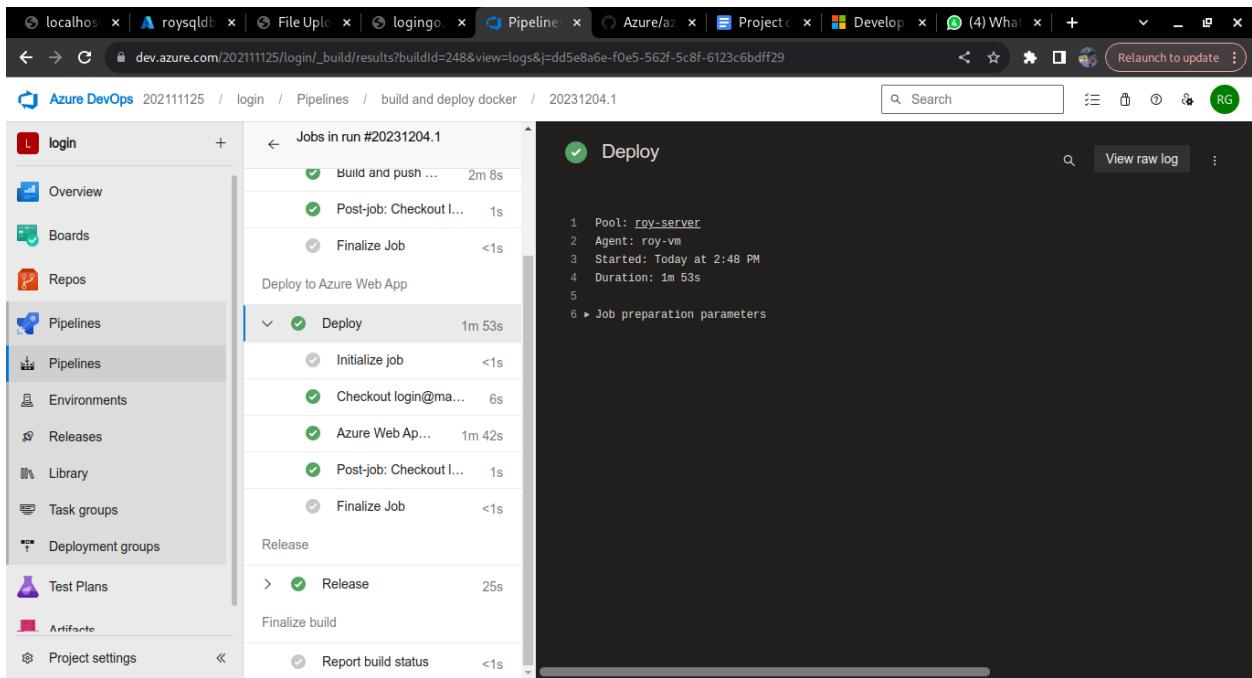
A screenshot of the Azure DevOps pipeline interface showing the build stage. The left sidebar is titled 'login' and includes 'Overview', 'Boards', 'Repos', 'Pipelines' (which is selected), 'Environments', 'Releases', 'Library', 'Task groups', 'Deployment groups', 'Test Plans', 'Artifacts', and 'Project settings'. The main area shows a list of jobs under 'Jobs in run #20231204.1'. The 'Build and push stage' section contains the following steps:

- Build (2m 20s):
 - Initialize job (<1s)
 - Checkout login@ma... 7s
 - Build and push ... 2m 8s
 - Post-job: Checkout l... 1s
 - Finalize Job (<1s)
- Deploy to Azure Web App:
 - Deploy 1m 53s
- Release:
 - Release 25s

The right panel displays the build summary with a green checkmark icon and the word 'Build'. The log output is as follows:

```
1 Pool: roy-server
2 Agent: roy-vm
3 Started: Today at 2:46 PM
4 Duration: 2m 20s
5
6 ▶ Job preparation parameters
```

Build stage



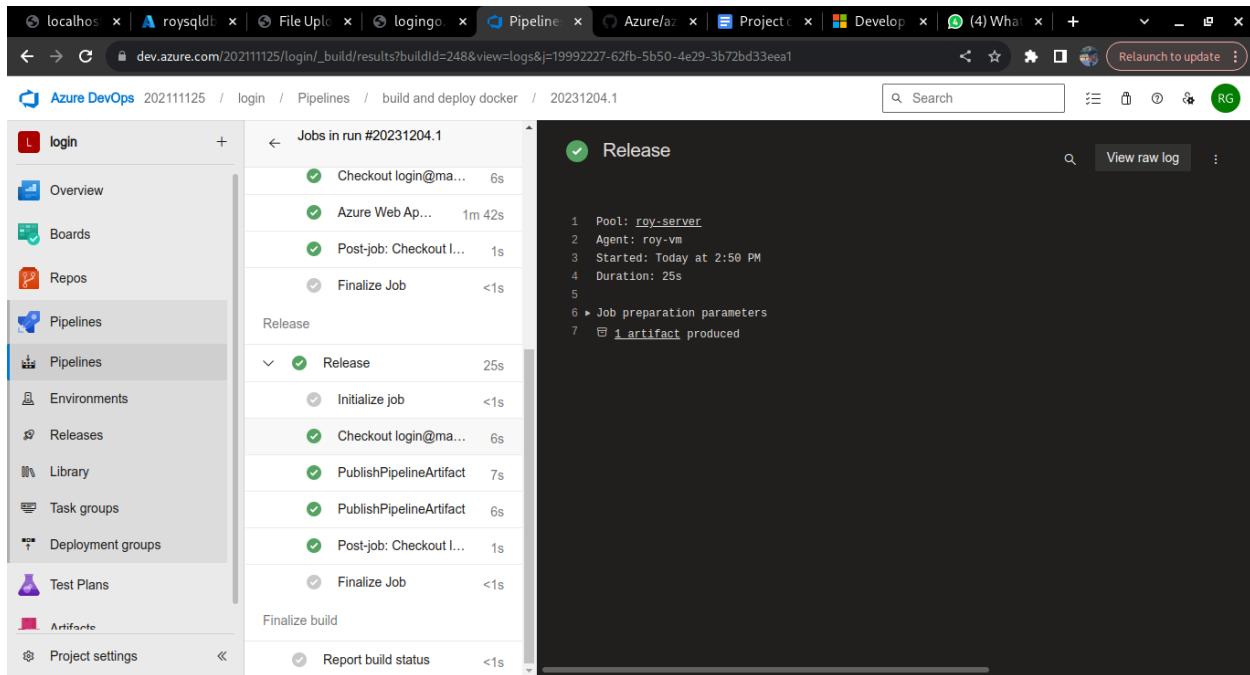
A screenshot of the Azure DevOps pipeline interface showing the deploy stage. The left sidebar is identical to the previous screenshot. The main area shows the same list of jobs under 'Jobs in run #20231204.1'. The 'Deploy' section contains the following steps:

- Deploy (1m 53s):
 - Initialize job (<1s)
 - Checkout login@ma... 6s
 - Azure Web Ap... 1m 42s
 - Post-job: Checkout l... 1s
 - Finalize Job (<1s)

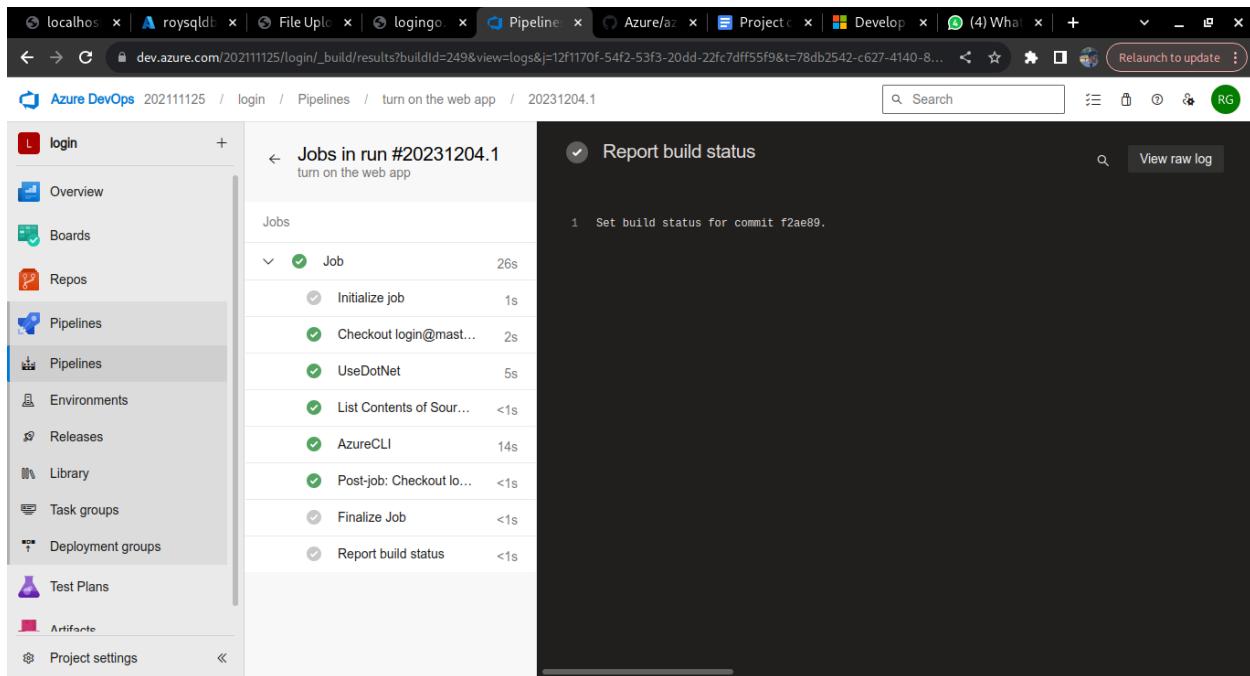
The right panel displays the deploy summary with a green checkmark icon and the word 'Deploy'. The log output is as follows:

```
1 Pool: roy-server
2 Agent: roy-vm
3 Started: Today at 2:48 PM
4 Duration: 1m 53s
5
6 ▶ Job preparation parameters
```

Deploy stage



Release stage



Here is another pipeline that turn on using bash scripts a the logingo web app

7 - Test case

The screenshot shows the Azure DevOps Test Plans interface. On the left, there is a sidebar with various project management and testing tools: login, Overview, Boards, Repos, Pipelines, Test Plans (selected), Test plans, Progress report, Runs, and Artifacts. Below the sidebar is a 'Project settings' link. The main area displays a 'Test Suites' section for 'login Team_Storie...' from Dec 3 - Dec 10, with a status of 'Current' and 0% run. A 'View report' button is available. To the right, a specific test point is shown under the heading '813 : having an api that inserts username and password to the sql database (ID: 817)'. The 'Execute' tab is selected, showing a 'Test Points (1 item)' list. The single item is titled 'testing it using testify framework in go' and is marked as 'In Progress'.

Creating the test plan for the post insertion into the db

A screenshot of the Azure DevOps interface showing a test case named "814 testing it using testify framework in go". The left sidebar shows navigation options like Boards, Work items, and Pipelines. The main area displays the test case details, including its state (Design), area (login), and iteration (loginSprint 1). The "Steps" section lists actions such as "log in the user" with expected results like "successfully logged in". The "Deployment" section provides instructions for tracking releases. The "Development" section includes a link to add an Azure Repos commit or pull request. The "Related Work" section is currently empty.

Creating all the steps into the test case

8 - KeyVault

A screenshot of the Microsoft Azure portal showing the "Key vaults" blade for the "keyvaultroy" resource. The left sidebar lists vault management options like Overview, Activity log, and Access control. The main pane displays the vault's essential details, including its Resource group ("roygebrayel-RG"), Location ("East US"), Subscription ID ("189af0e7-fa98-4751-b5a8-0d39f4d49d5f"), and Directory ID ("e731378a-11b1-405c-bb1c-c047ae5a5d54"). It also shows the vault's URI ("https://keyvaultroy.vault.azure.net/"), Sku ("Standard"), and other properties like Tags, Purge protection, and Soft-delete. A "Get started" button is at the bottom.

This is the keyvault configuration

key vaults

keyvaultroy | Secrets

Name	Type	Status	Expiration date
storageacoountsecretconnec...		✓ Enabled	
functionappkey	default api	✓ Enabled	
sqlldbpassword		✓ Enabled	

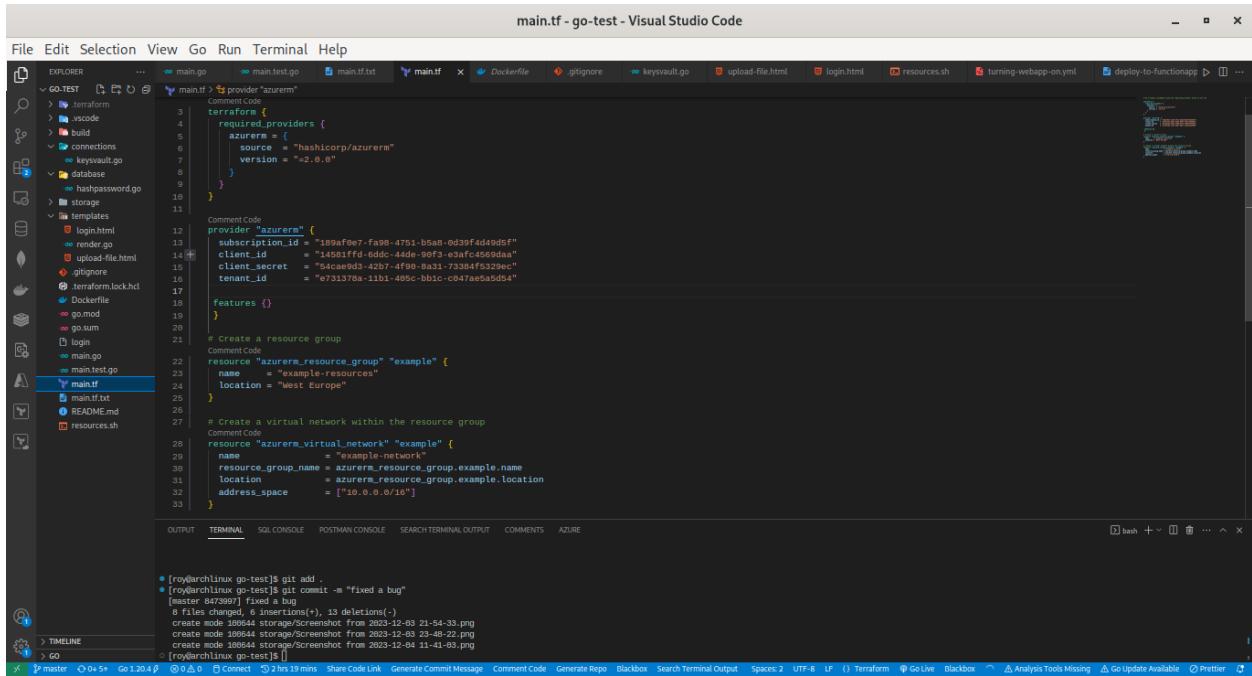
Those are the secrets created

functionnodeapp | Configuration

Name	Value	Source
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to show value	App Service
AzureWebJobsStorage	Hidden value. Click to show value	App Service
connection	Hidden value. Click to show value	Key vault Reference
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to show value	App Service
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to show value	App Service
WEBSITE_CONTENTAZUREFILECONNECTION	Hidden value. Click to show value	App Service
WEBSITE_CONTENTSHARE	Hidden value. Click to show value	App Service
WEBSITE_ENABLE_SYNC_UPDATE_SITE	Hidden value. Click to show value	App Service
WEBSITE_NODE_DEFAULT_VERSION	Hidden value. Click to show value	App Service
WEBSITE_RUN_FROM_PACKAGE	Hidden value. Click to show value	App Service

And here is where i created the connection and gived it the secret key value i just created

9 - terraform



The screenshot shows the Visual Studio Code interface with the title bar "main.tf - go-test - Visual Studio Code". The code editor displays the `main.tf` file, which contains Terraform configuration code. The code defines an `azurerm` provider and creates a resource group named "example" in the "West Europe" location. It also creates a virtual network named "example-network" with a single address space of `[10.0.0.0/16]`. The code editor has syntax highlighting for Terraform and includes comments explaining each section. Below the code editor, the status bar shows the terminal command `[royarchlinux go-test] git add .` and other git commit history.

```
provider "azurerm" {
    required_providers {
        azurerm = {
            source  = "hashicorp/azurerm"
            version = "=2.0.0"
        }
    }

    provider "azurerm" {
        id      = "10af0e7f-fa99-4751-b5a0-0d39f4d4965f"
        client_id = "158afffd-6ddc-44de-9bf3-e3afc4569daa"
        client_secret = "54caed3d-42b7-4f98-8a31-733bf5329ec"
        tenant_id = "e731378a-11b1-405c-bb1c-c847ae5a5d54"
    }

    features {}

    # Create a resource group
    resource "azurerm_resource_group" "example" {
        name     = "example-resources"
        location = "West Europe"
    }

    # Create a virtual network within the resource group
    resource "azurerm_virtual_network" "example" {
        name           = "example-network"
        resource_group_name = azurerm_resource_group.example.name
        location       = azurerm_resource_group.example.location
        address_space  = ["10.0.0.0/16"]
    }
}
```

This is the `main.tf` file that automatically turns on all the written resources in azure using the `azurerm` provider

```

roy@archlinux:~ 
The host "registry.terraform.io" given in provider source address "registry.terraform.io/hashicorp/azurerm" does not offer a Terraform provider registry.

[roy@archlinux go-test]$ terraform init -upgrade
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "2.0.0"...
- Using previously-installed hashicorp/azurerm v2.0.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

[roy@archlinux go-test]$ terraform apply

  Error: Error building account: Error getting authenticated object ID: Error listing Service Principals: autorest.DetailedError{Original:adal.tokenRefreshError{message:"adal: Refresh request failed. Status Code = '401'. Response body: (\\"error\\":\\"invalid_client\\",\\"error_description\\":\\"AADSTS7000215: Invalid client secret provided. Ensure the secret being sent in the request is the client secret value, not the client secret ID, for a secret added to app '14581ffd-6ddc-44de-90f3-e3afc4569daa'. Trace ID: 55bc7685-b684-423d-aafa-eb3ab5751300 Correlation ID: 2d2890ec-8adc-4384-8155-c631d8e0221e Timestamp: 2023-12-04 13:17:33Z\\",\\"error_codes\\":\[7000215],\\"timestamp\\":\\"2023-12-04 13:17:33Z\\",\\"trace_id\\":\\"55bc7685-b684-423d-aafa-eb3ab5751300\\",\\"correlation_id\\":\\"2d2890ec-8adc-4384-8155-c631d8e0221e\\",\\"error_uri\\":\\"https://Login.microsoftonline.com/error?code=7000215\\", resp:(*http.Response)(0xc000462900)}, PackageType:"azure.BearerAuthorizer", Method:"WithAuthorization", StatusCode:401, Message:"Failed to refresh the Token for request to https://graph.windows.net/e731378a-1b1-405c-bb1c-c047ae5a5d54/servicePrincipals?%24filter=appId+eq+%2714581ffd-6ddc-44de-90f3-e3afc4569daa%27&api-version=1.6", ServiceError:[]uint8(nil), Response:(*http.Response)(0xc000462900)}}

      with provider["registry.terraform.io/hashicorp/azurerm"],
      on main.tf line 12, in provider "azurerm":
      12: provider "azurerm" {}

[roy@archlinux go-test]$ az account show

```

Running the terraform :

Conclusion

In conclusion, this project fulfills the academic requirements aimed in this course , and i'm confident to say that i've quite understand how the cloud and the devops world works by implementing real case problems and managing them

Appendix

Github repo : [logingo](#)

Azure repo : [logingo](#)

This project was made from 26/11/23 till 5/12/23 and submitted in 7/12/23

