

AutoOps

Team Members
CSCE 5214.005
University of North Texas

Mounika Ponnamp (11610822)
Sai Sarat Chandra Vytla (11588541)
Rahul Manikonda (11608670)

GitHub Link for the AutoOps Project : [sdaiproject/SDAI-Project-Final](https://github.com/sdaiproject/SDAI-Project-Final)
github.com

1. Project Introduction

1.1 Motivation:

In the cloud industry buzzwords like IaaS, DevOps, AIOps, etc. have risen to prominence. In AutoOps we have cohered these cutting-edge technologies and tools to provide one-stop solution for scalability, operationality and bring agility with automation and intelligence in modern cloud data platforms. AutoOps utilizes technologies like Terraform templates, Azure Resource manager (ARM) templates, Azure native services like Azure Active Directory, Azure ML workspace, advanced AI/ML algorithms, service management tools, etc. To auto deploy and provision Azure data platform services, along with monitoring and validation of resource deployment. The idea behind the selection of open-source technologies (Terraform, Azure CLI, etc.) and Azure native services is to reduce upfront cost and investment from customers on tools and technologies. It also helps our customers to reduce go to market time, increasing efficiency, productivity and operationality.

1.2 Significance :

AutoOps helps people to provision their cloud platform which contains services like Azure storage account, Azure function apps, Azure logic apps, etc. It reduces the deployment time as we are using automation tools like Terraform. Terraform is useful to deploy all the Azure services with a single command. All we need to do is to collect Azure service templates for the deployment, consolidate all the templates and within hours of time Azure data platform can be deployed in whichever environment you want. Terraform makes our lives much easier because this automates the tedious tasks. If we deploy Azure services manually, it will take days or weeks to just provision a few services. On top of that, including security features takes lots of time. But with the help of automation we can get these advanced features within a few hours of time like data encryption, masking.

1.3 Objective :

The main motto of developing this project is to reduce the time to deploy Azure services like Azure data factory, Azure Synapse Analytics, Azure data lake, Azure Data bricks, etc. Usually people waste lots of time in provisioning these Azure services. Instead they can focus on main tasks like creating Azure data factory pipelines, performing advanced transformations on pipelines which are very crucial for business needs. And developers find it difficult to monitor

the deployed platform, So If we could collect all azure data platform logs when some abnormal accident happens,It would be very helpful for them. So we can use Log analytics works which is a generic repository to store all azure data platform logs to make developer work easier. After knowing the exact root cause of the anomaly, we should get the proper resolution steps to tackle that problem. So we decided to build an AI/ML engine to get the proper resolution to resolve the issues that occur in azure data platform.

1.4 Features :

- **Deployment** : The beauty of the AutoOps project is it saves lots of time for the people as they do not need to wait for days or weeks for Azure data service provisioning. With the help of terraform, without any human intervention, we can have our platform ready within hours of time. This makes the project very unique.
- **Monitoring**: Azure offers customisable diagnostic options for every single service. This will result in the collection of all service logs into what is known as an Azure log analytics workspace. This enables the developer to focus more on the functionality while making it easier for them to examine the specifics of the anomaly.
- **Anomaly collection** : This the additional advantage for the AutoOps project,as developers do not need to go for any third party applications for storing error details from Azure data platform. We can use azure native service “Log analytics Workspace” for collecting all log and metric details in azure data platform.This is also a cost effective service.
- **AI/AM Engine** : we train the model with all the common error messages with their corresponding resolutions. After our model is ready with better accuracy, we will publish our model into Azure ML workspace. This will give us proper resolution steps which will help us to solve the issue in Azure data platform.
- **Resolution** : After getting the resolution steps from the AI/ML engine, If it is a common issue, In order to solve the issue we can have some kind of document to help the developer solve the issue easily or we can have automatic resolution which will auto resolve the problems.

2. Related Work (Background) :

The same concept “AutoOps” is developed in the multinational companies like Tata consultancy services(TCS), CISCO,etc.. They explained in detail about the AutoOps in their vision. Gartner, a world - wide consulting and research firm, invented the term AutoOps. According to Gartner, AutoOps "consolidates machine learning and big data for automating IT operations procedures such as causality determination, anomaly detection, event correlation”.

To implement and operate AutoOps, organizations must line up various components of software and hardware including ML and AI engines and specially trained data centers, in addition to expertise to deliver. Numerous providers provide AutoOps solutions that incorporate ML, AI and big data functionalities. These applications automate and enhance incident surveillance, managed services, and other functions. These solutions are commonly referred to as AutoOps operating systems by most providers. More than anything, AutoOps is a strategy for reshaping IT processes across the board.

A company cannot implement AutoOps without the capability to connect its IT systems so that they can exchange data and benefit from one another. Systems integration prompts an open application programming interface (API), which means that the product company produces the API freely accessible to software engineers. Software development kits (SDKs) are indeed required for AutoOps implementation. These toolkits are used by developers to develop custom applications which can be incorporated to or attached towards other initiatives.

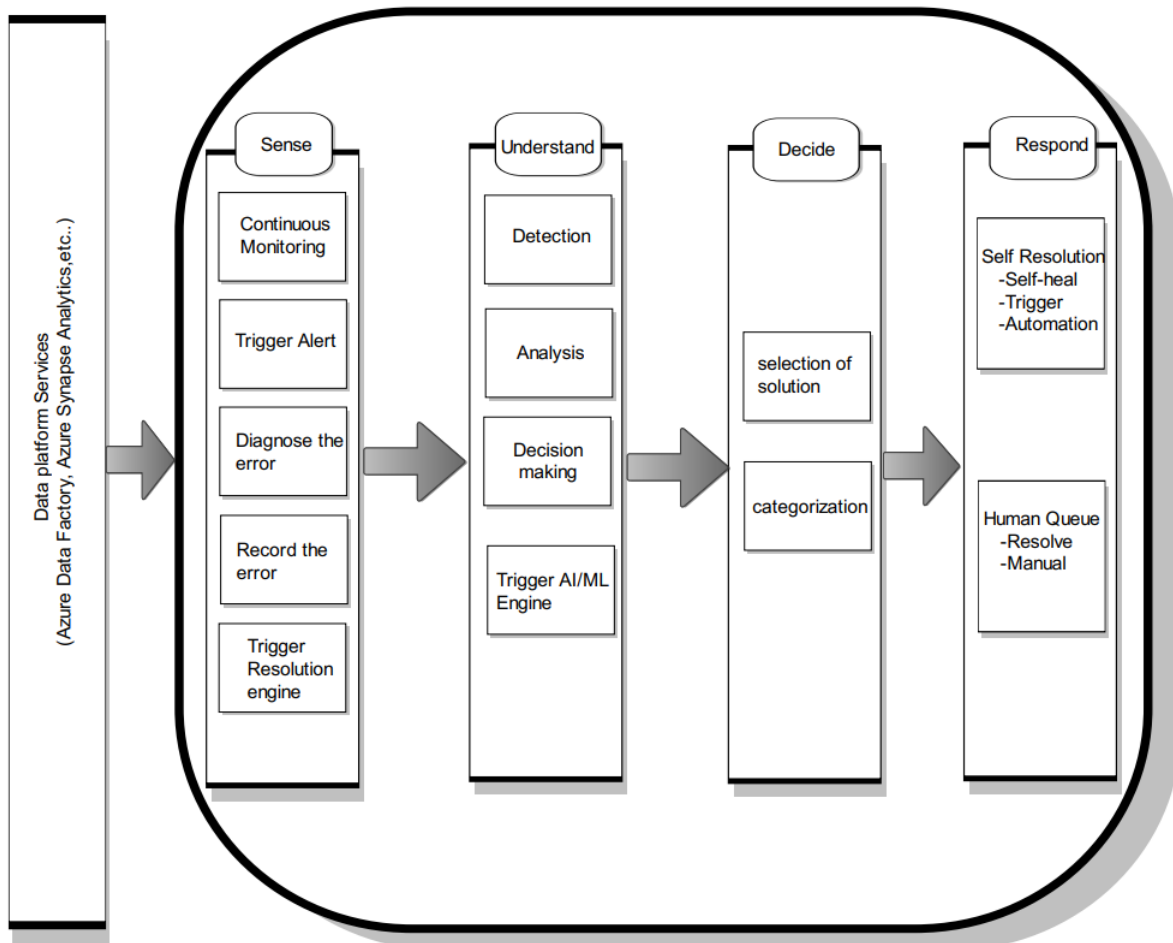
3. Dataset :

The dataset that we used for our model contains two important columns Error Message and Resolution. Error messages are the exact anomaly details we receive from log analytics workspace after collecting from Azure data platform. Resolutions are the steps to resolve the issues that occur in Azure data platform. There are plenty of datasets available in open source websites like Kaggle,etc. I have taken the data from official Microsoft documents as our dataset for our model.

The url for the dataset : [SDAI-Project-Final/Dataset.csv at main · sdaiproject/SDAI-Project-Final \(github.com\)](https://github.com/sdaiproject/SDAI-Project-Final/blob/master/Dataset.csv)

4. Detail design of Features :

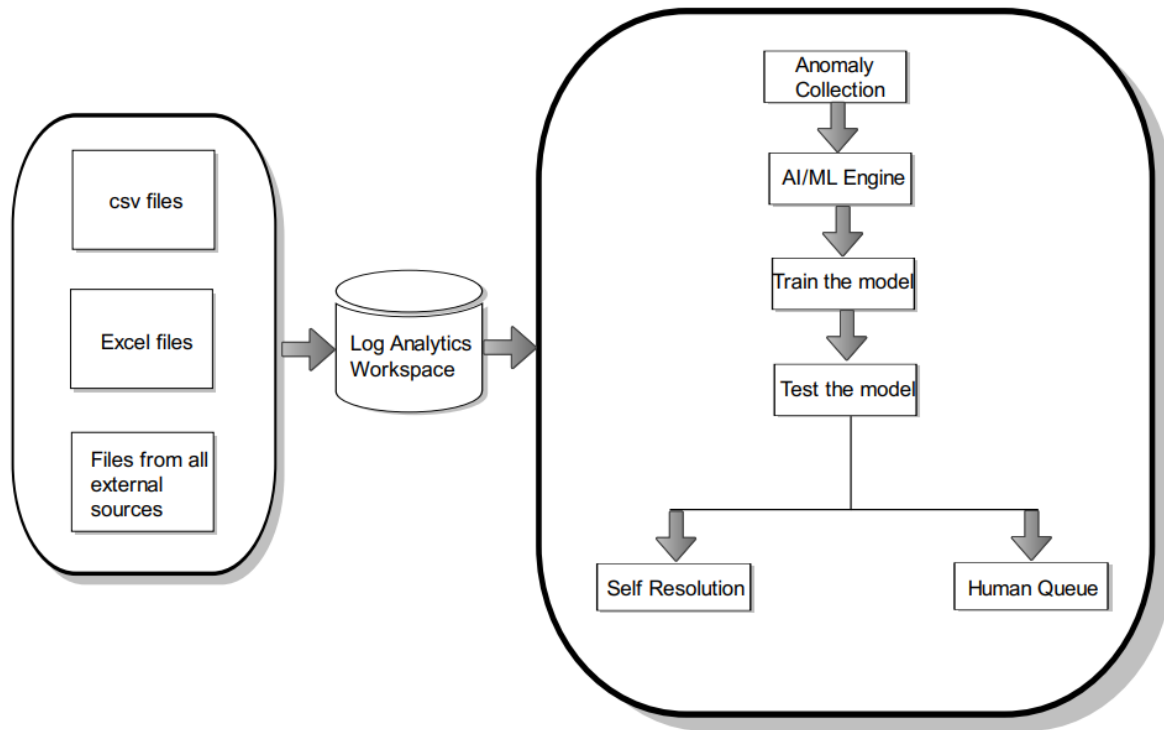
AutoOps Architecture :



The above diagram shows the entire architecture and model of the AutoOps. First of all we deploy the azure services (Azure Data bricks, azure synapse analytics, azure data lake, function apps, web apps,etc..) onto the azure platform automatically without human need by using terraform tools. AutoOps tracks all the azure services activities. It will check the health of the services like (azure data factory,azure data bricks, synapse analytics,etc..) and does continuous monitoring. If any abnormality occurs in the Azure data platform, it will trigger and all the error log details will be sent to Azure log analytics workspace. After detecting the error, AutoOps analyzes whether it can be resolved or if it has proper resolution steps using the AI/ML engine which makes use of advanced machine learning algorithms. It will be categorized into two types: self-Resolution and Human-Queue. If the AI/ML engine has resolution steps for that error,it will come under self resolution. If it is an unknown error then it

will be sent to the human queue, which means the technical support team has to take care of the issue.

Flow Diagram:



The above diagram shows the workflow of how AutoOps works. In the first step, AutoOps collects all the error logs that occur in the azure data platform. It will collect the logs in the form of csv,excel or any other forms. It will be sent to the Log analytics workspace. Since it automatically detects the error and collects logs, we call it automatic anomaly detection. Then we send this anomaly to the AI/ML engine which will provide us proper resolution steps to resolve the issue. If the AI/ML engine has a solution,it will send back the resolution steps, otherwise human forces need to come and fix the issue manually.

5. Analysis:

AutoOps assists people in provisioning their cloud platform, which includes services such as Azure storage accounts, Azure function apps, Azure logic apps, and so on. It decreases deployment time because we use automation tools such as Terraform. Terraform is beneficial for quickly deploying all Azure services with a single command. All we need to do is collect Azure service templates for deployment, consolidate all the templates, and the Azure data platform can be deployed in any environment within hours. Terraform simplifies our lives by automating time-consuming tasks. Manually deploying Azure services will take days or weeks just to provision a few services. Furthermore, including security features takes a significant amount of time.

The major goal of creating this project is to shorten the time it takes to install Azure services such as Azure data factory, Azure synapse analytics, Azure data lake, Azure Data bricks, and so on. Humans actually waste so much time provisioning many such Azure services. Rather than, they can concentrate on crucial functions such as creating Azure Data Factory pipelines and performing advanced transformations on pipelines that are essential for economic demands. And programmers are finding it challenging to track the installed platform, so it would be very helpful if we were able to collect all azure data platform logs when an anomalous incident occurs. So, to make developer work easier, we can use Log analytics works, as it is a standard archive for storing all Azure Data Platform logs. Humans should be given the proper resolution steps to address the issue once we know the exact root cause of the anomaly. As a result, we decided to build an AI/ML engine to get the proper resolution to the issues that arise in the Azure Data Platform.

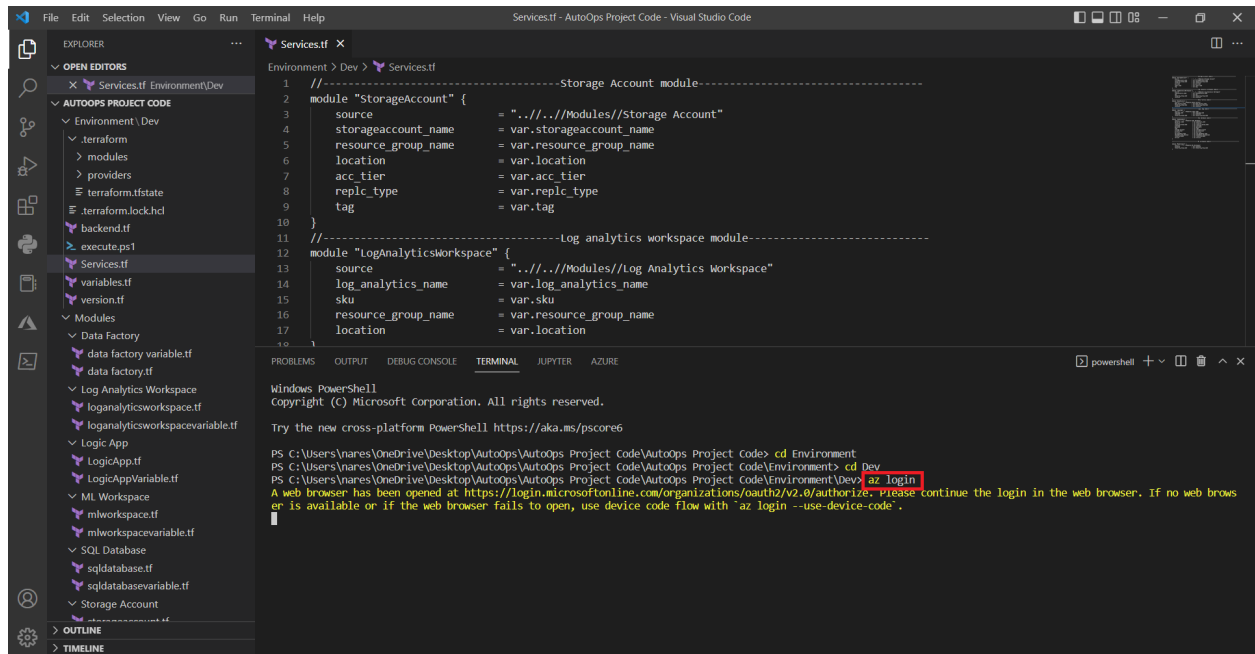
6. Implementation

Prerequisites that needs to be present to build our project and for smooth functioning are explained below:

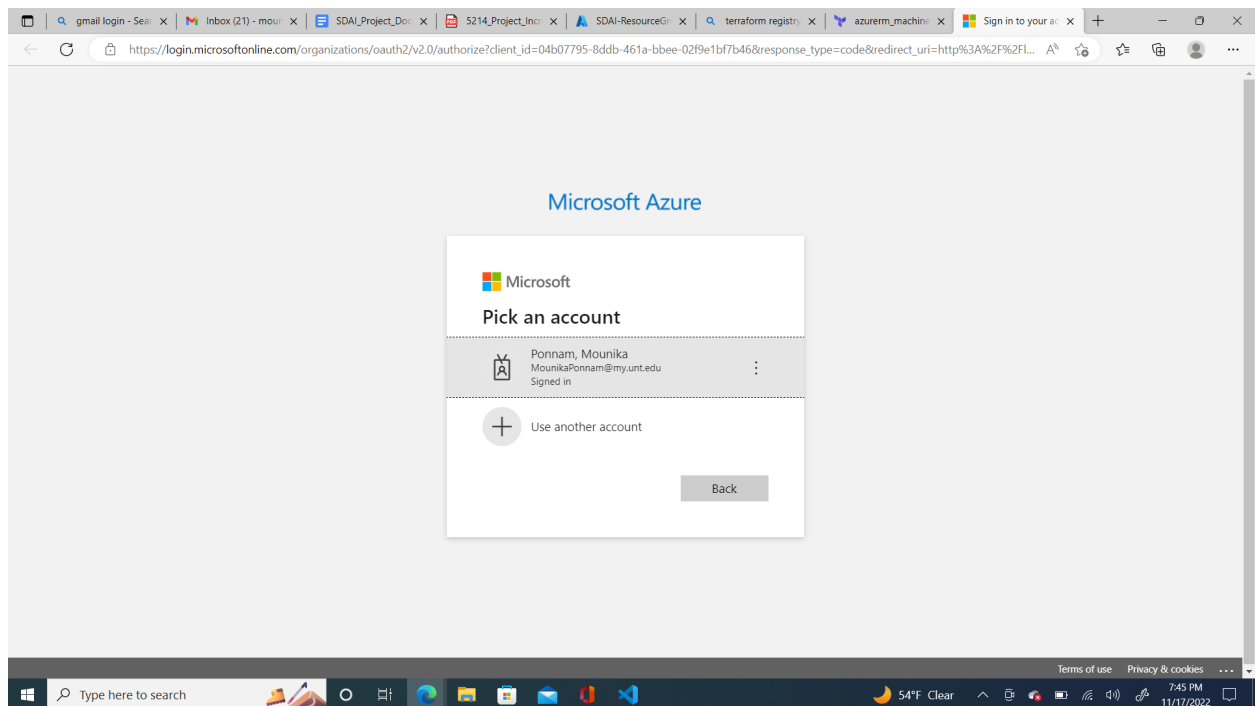
- Terraform latest version
- Microsoft Azure active account
- Azure CLI : To run terraform command for deploying azure cloud services like storage account,data factory,synapse analytics,etc..
- IDEs like Visual Studio Code or Visual Studio 2022 are required to use terraform tool
- For version control and to manage the project code folder and files deployment we need GitHub.

Deployment Feature:

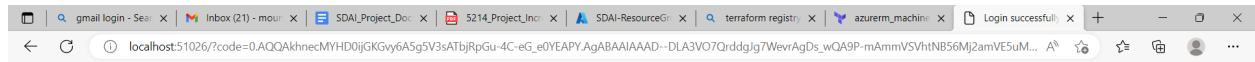
Step 1: First to login to our azure portal we need to run command “az login”



Step 2 : It will navigate to the azure portal in default browser

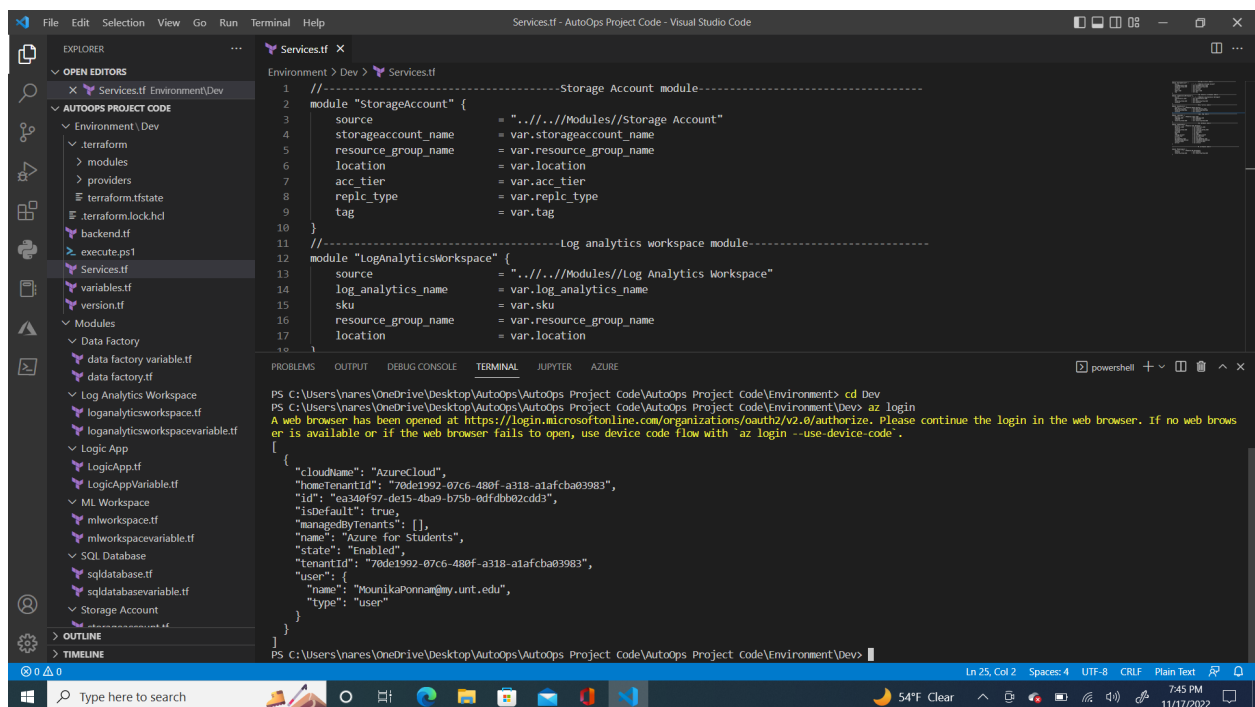


Step 3 : If the login is successful, we will get below message

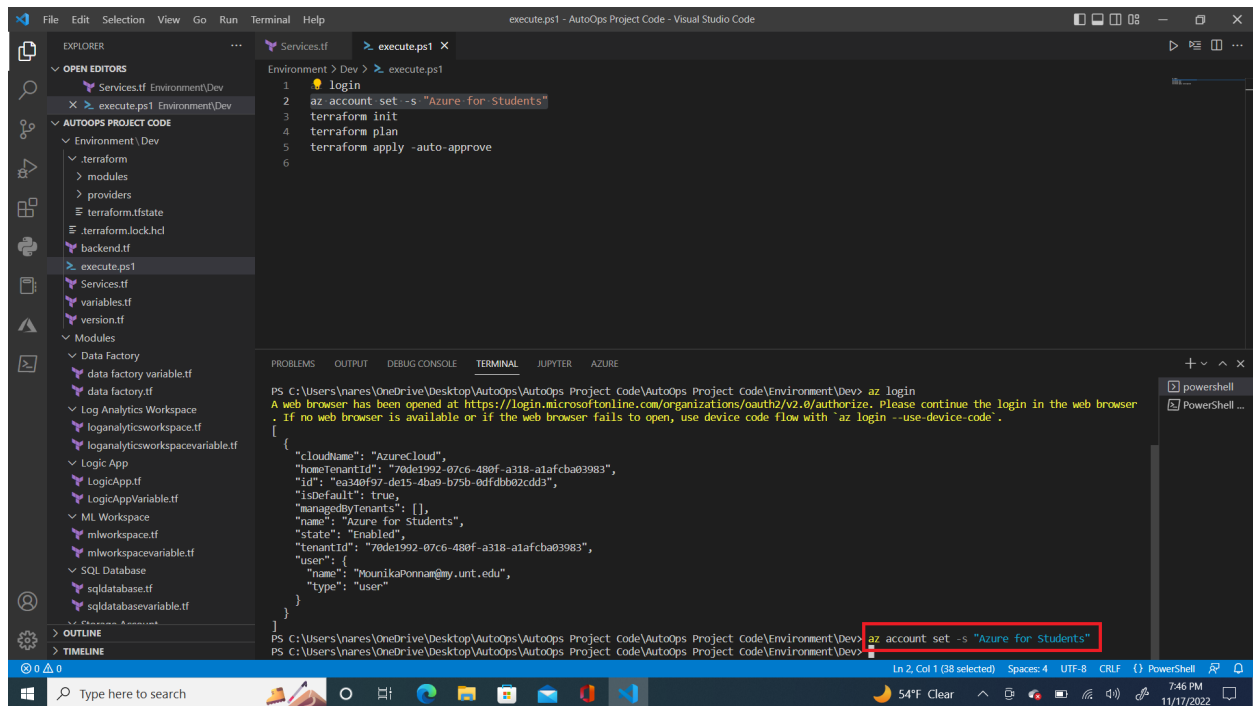


You have logged into Microsoft Azure!

You can close this window, or we will redirect you to the [Azure CLI documentation](#) in 10 seconds.



Step 4 : Then we need to set the subscription

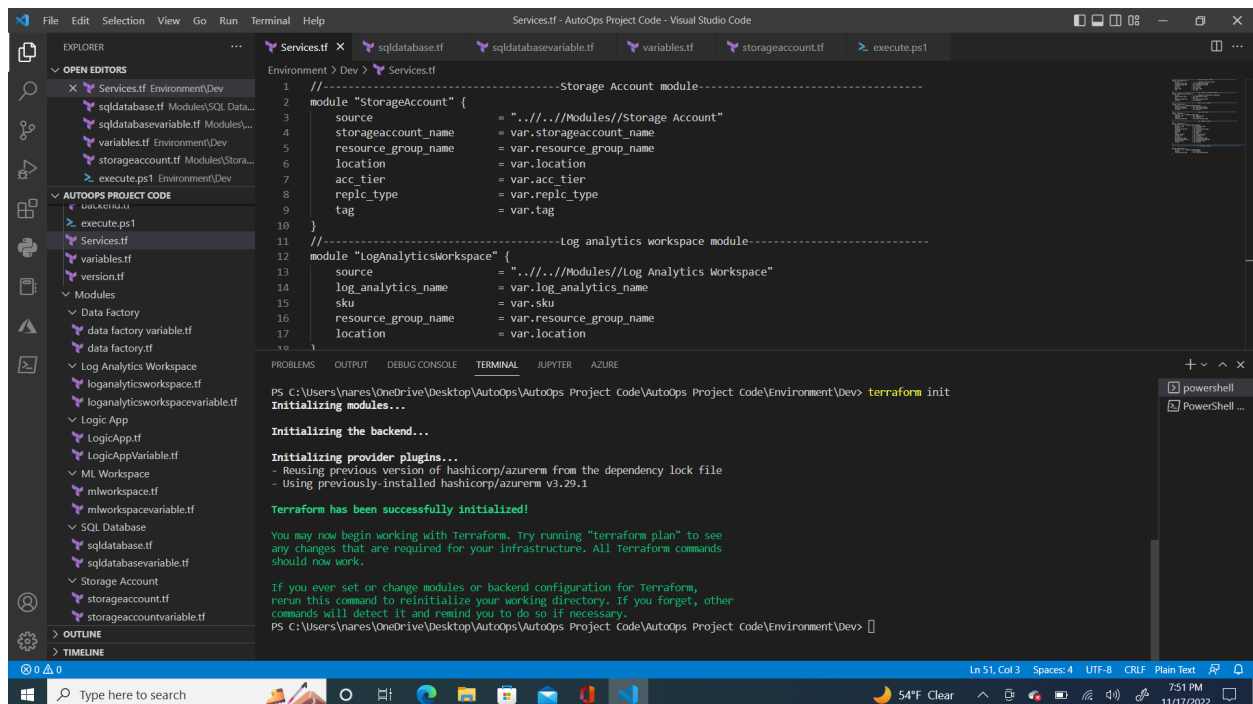


The screenshot shows the Visual Studio Code interface with the 'execute.ps1' file open in the editor. The file contains a list of commands to be executed in a PowerShell terminal. The command 'az account set -s "Azure for Students"' is highlighted in the editor. The terminal output shows the command being executed and the resulting JSON output for the 'az login' command.

```
Environment > Dev > execute.ps1
1 login
2 az account set -s "Azure for Students"
3 terraform init
4 terraform plan
5 terraform apply -auto-approve
6

PS C:\Users\Nares\OneDrive\Desktop\AutoOps\AutoOps Project Code\AutoOps Project Code\Environment\Dev> az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser
. If no web browser is available or if the web browser fails to open, use device code flow with 'az login --use-device-code'.
{
  "cloudName": "AzureCloud",
  "homeTenantId": "70de1992-07c6-480f-a318-a1afcba03983",
  "id": "ea340f97-de15-4ba9-b75b-0dfdbb02cdd3",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Azure for Students",
  "state": "Enabled",
  "tenantId": "70de1992-07c6-480f-a318-a1afcba03983",
  "user": {
    "name": "MounikaPonnangmy.unt.edu",
    "type": "user"
  }
}
```

Step 5 : In Order to initialize terraform to automatically provision the resources without human intervention, we need to run “**terraform init**” command



The screenshot shows the Visual Studio Code interface with the 'execute.ps1' file open in the editor. The file contains a list of commands to be executed in a PowerShell terminal. The command 'terraform init' is highlighted in the editor. The terminal output shows the command being executed and the resulting output for the 'terraform init' command.

```
Environment > Dev > Services.tf
1 //-----Storage Account module-----
2 module "StorageAccount" {
3   source = "../../Modules/Storage Account"
4   storageaccount_name = var.storageaccount_name
5   resource_group_name = var.resource_group_name
6   location = var.location
7   acc_tier = var.acc_tier
8   replc_type = var.replc_type
9   tag = var.tag
10 }
11 //-----Log Analytics workspace module-----
12 module "LogAnalyticsworkspace" {
13   source = "../../Modules/Log Analytics Workspace"
14   log_analytics_name = var.log_analytics_name
15   sku = var.sku
16   resource_group_name = var.resource_group_name
17   location = var.location
18 }

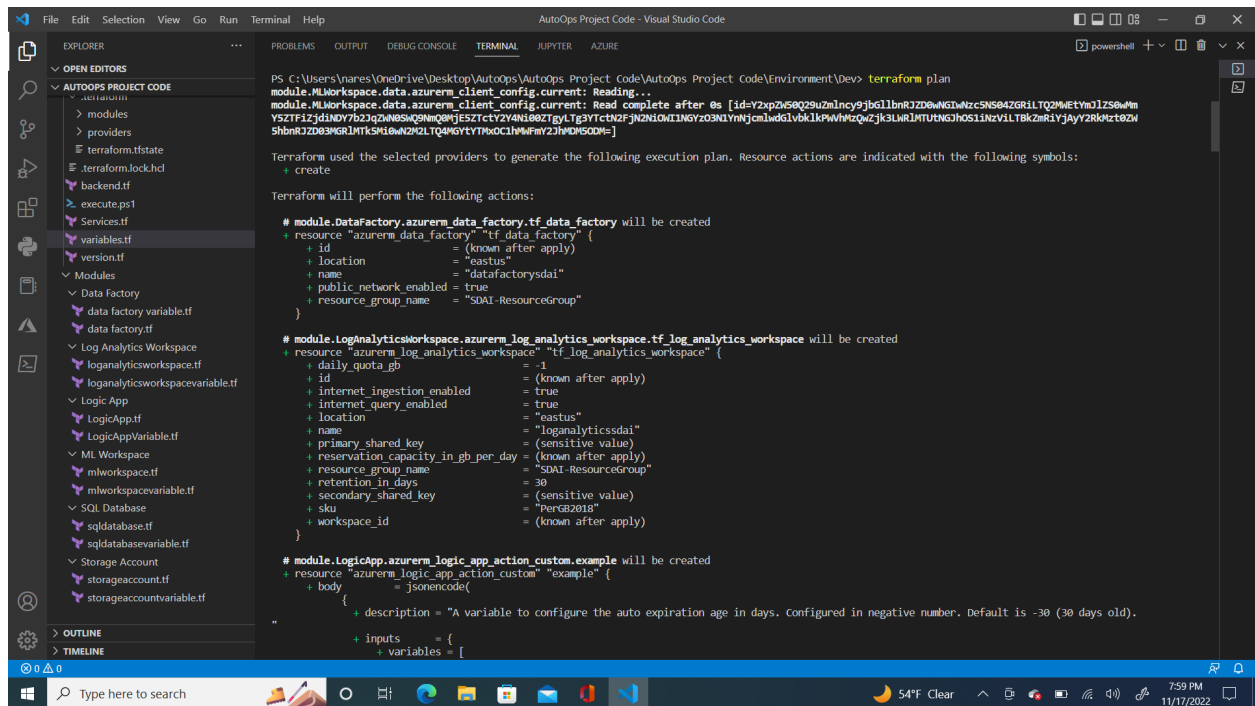
PS C:\Users\Nares\OneDrive\Desktop\AutoOps\AutoOps Project Code\AutoOps Project Code\Environment\Dev> terraform init
Initializing modules...
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Using previously-installed hashicorp/azurerm v3.29.1

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.
PS C:\Users\Nares\OneDrive\Desktop\AutoOps\AutoOps Project Code\AutoOps Project Code\Environment\Dev> []
```

Step 6 : To make sure terraform is deploying the resources correctly, we can run “**terraform plan**” command



```
PS C:\Users\vnanes\OneDrive\Desktop\AutoOps\AutoOps Project Code\AutoOps Project Code\Environment\Dev> terraform plan
module.MLWorkspace.data.azurem_client_config.current: Reading...
module.MLWorkspace.data.azurem_client_config.current: Read complete after 0s [id=Y2xpZm50Q29uZmlncy9jbGllbnR3ZD9wNGJhZC5NS04ZGRlLTQ2MmEYmDZS0wMm
YSZTFiZjdiNDY7b2JqZmN0S09mN0MjESZTctY2Y4Ni00ZTgyLTg3YTctN2FjN2NiOW11NGYzO3NiYmNjcmVwdGlvbk1kPWNhZGZjLjE3LmR1MTU5NGJOS11mZVl1T8kZmR1YjY2Rk9kZmR1
ShbnR3ZD9wNGRlMTk5M10wN2M2LTQ4NGYyYmN0C1h4MFAwZ2M2MDM0O0=]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

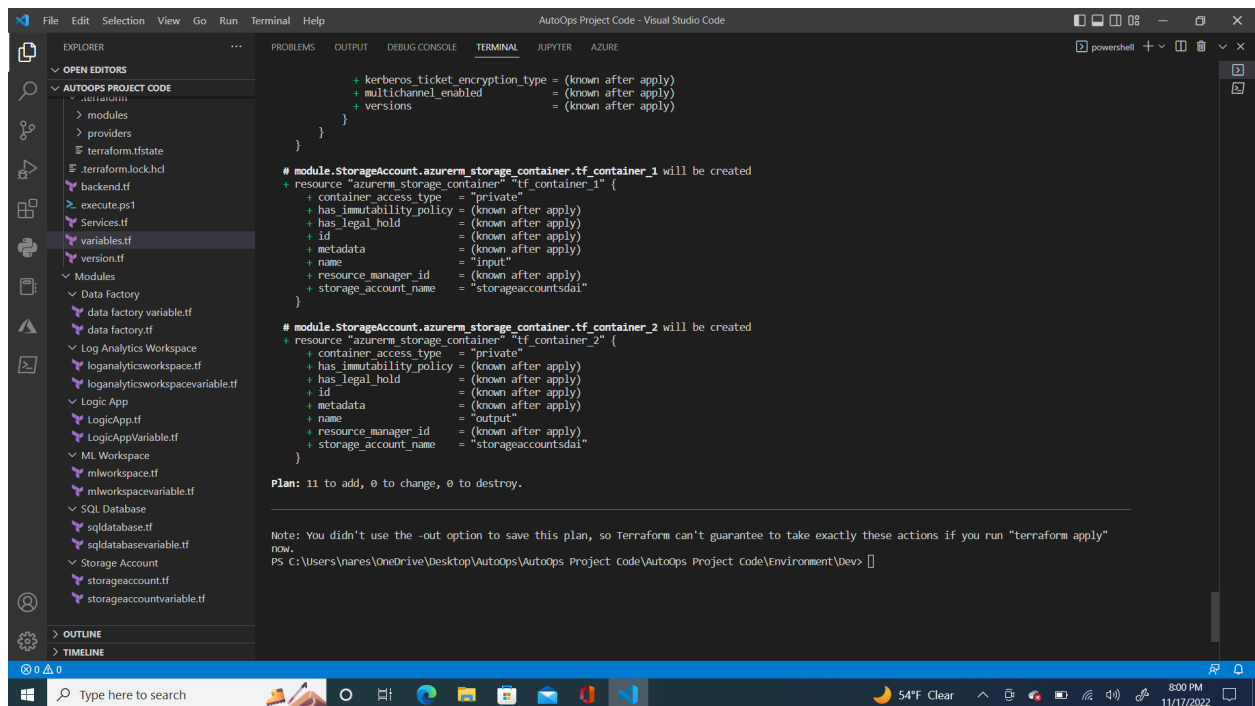
# module.DataFactory.azurem_data_factory.tf_data_factory will be created
+ resource "azurem_data_factory" "tf_data_factory" {
  + id               = (known after apply)
  + location         = "eastus"
  + name            = "datafactorysdai"
  + public_network_enabled = true
  + resource_group_name = "SDAI-ResourceGroup"
}

# module.LogAnalyticsWorkspace.azurem_log_analytics_workspace.tf_log_analytics_workspace will be created
+ resource "azurem_log_analytics_workspace" "tf_log_analytics_workspace" {
  + daily_quota_gb = 1
  + id             = (known after apply)
  + internet_ingestion_enabled = true
  + internet_query_enabled = true
  + location        = "eastus"
  + name           = "loganalyticssdai"
  + primary_shared_key = (sensitive value)
  + reservation_capacity_in_gb_per_day = (known after apply)
  + resource_group_name = "SDAI-ResourceGroup"
  + retention_in_days = 30
  + secondary_shared_key = (sensitive value)
  + sku               = "PerGB2018"
  + workspace_id      = (known after apply)
}

# module.LogicApp.azurem_logic_app_action_custom.example will be created
+ resource "azurem_logic_app_action_custom" "example" {
  + body = jsonencode(
    {
      + description = "A variable to configure the auto expiration age in days. Configured in negative number. Default is -30 (30 days old)."
    }
  )
  + inputs = {
    + variables = [

```

Step 7 : If the configuration is correct, then we will get confirmation message as shown in below image



```
+ kerberos_ticket_encryption_type = (known after apply)
+ multichannel_enabled = (known after apply)
+ versions = (known after apply)
}
}

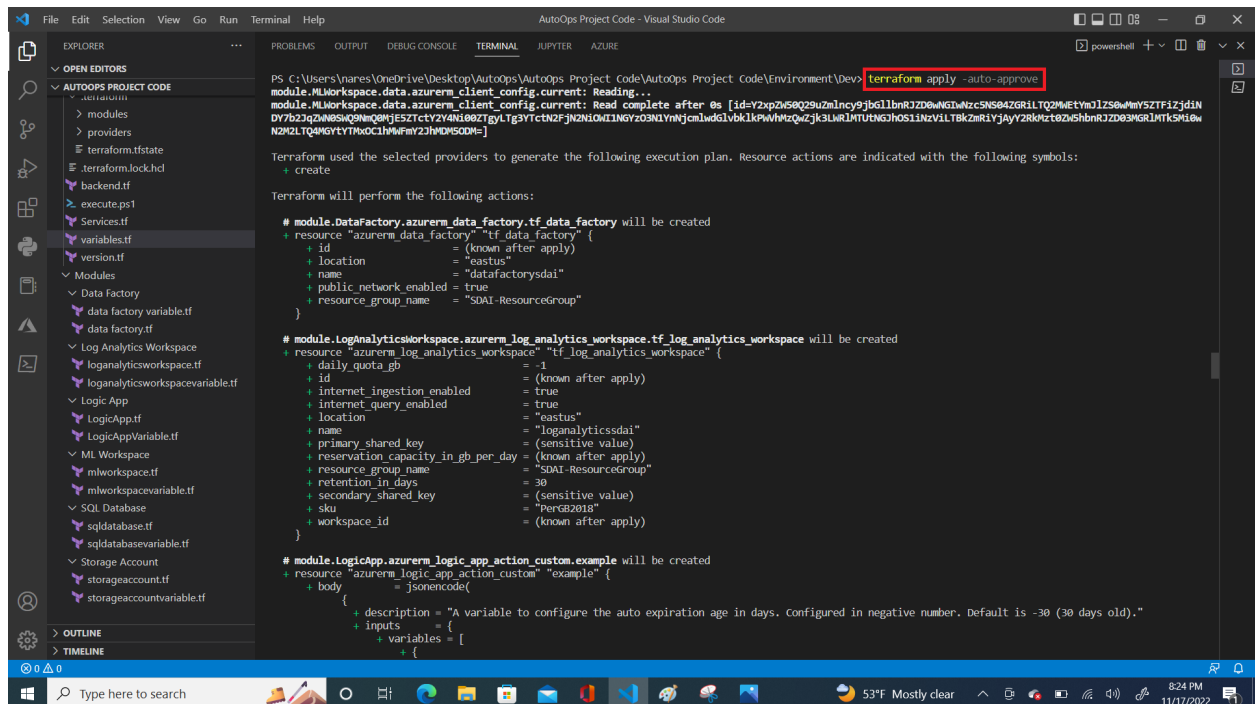
# module.StorageAccount.azurem_storage_container.tf_container_1 will be created
+ resource "azurem_storage_container" "tf_container_1" {
  + container_access_type = "private"
  + has_immutability_policy = (known after apply)
  + has_legal_hold = (known after apply)
  + id = (known after apply)
  + metadata = (known after apply)
  + name = "input"
  + resource_manager_id = (known after apply)
  + storage_account_name = "storageaccountsai"
}

# module.StorageAccount.azurem_storage_container.tf_container_2 will be created
+ resource "azurem_storage_container" "tf_container_2" {
  + container_access_type = "private"
  + has_immutability_policy = (known after apply)
  + has_legal_hold = (known after apply)
  + id = (known after apply)
  + metadata = (known after apply)
  + name = "output"
  + resource_manager_id = (known after apply)
  + storage_account_name = "storageaccountsai"
}

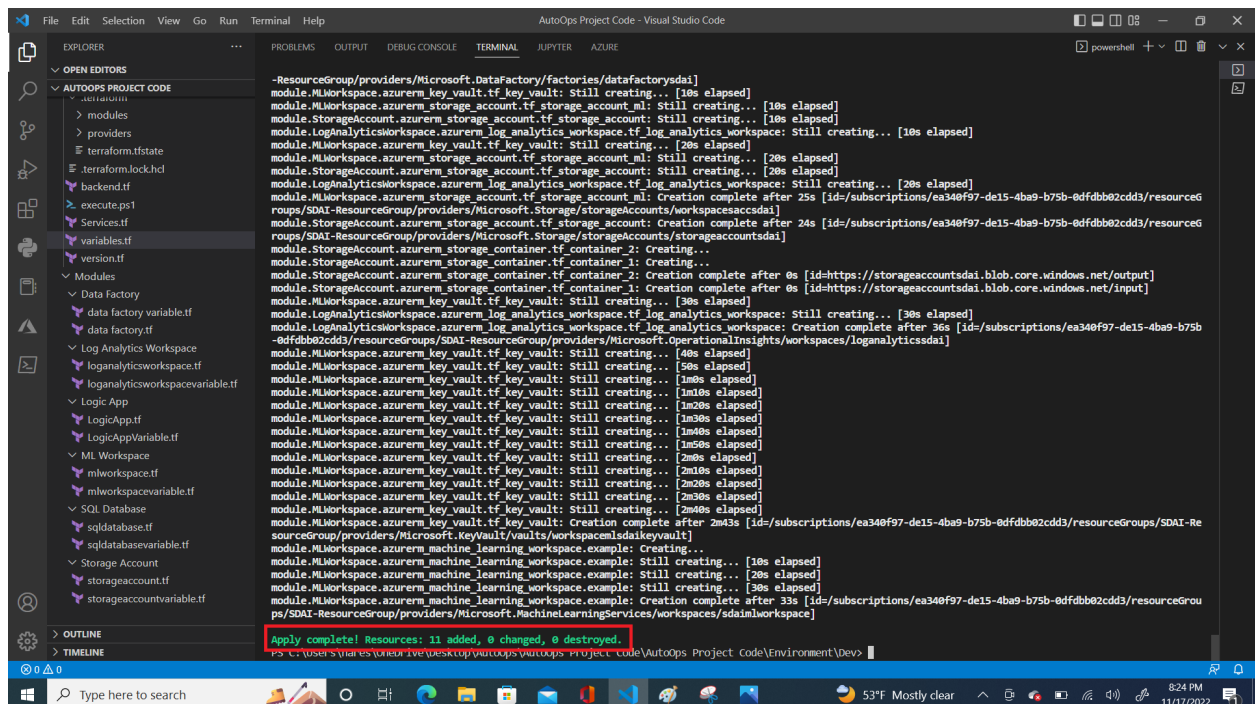
Plan: 11 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply"
now.
PS C:\Users\vnanes\OneDrive\Desktop\AutoOps\AutoOps Project Code\AutoOps Project Code\Environment\Dev> []
```

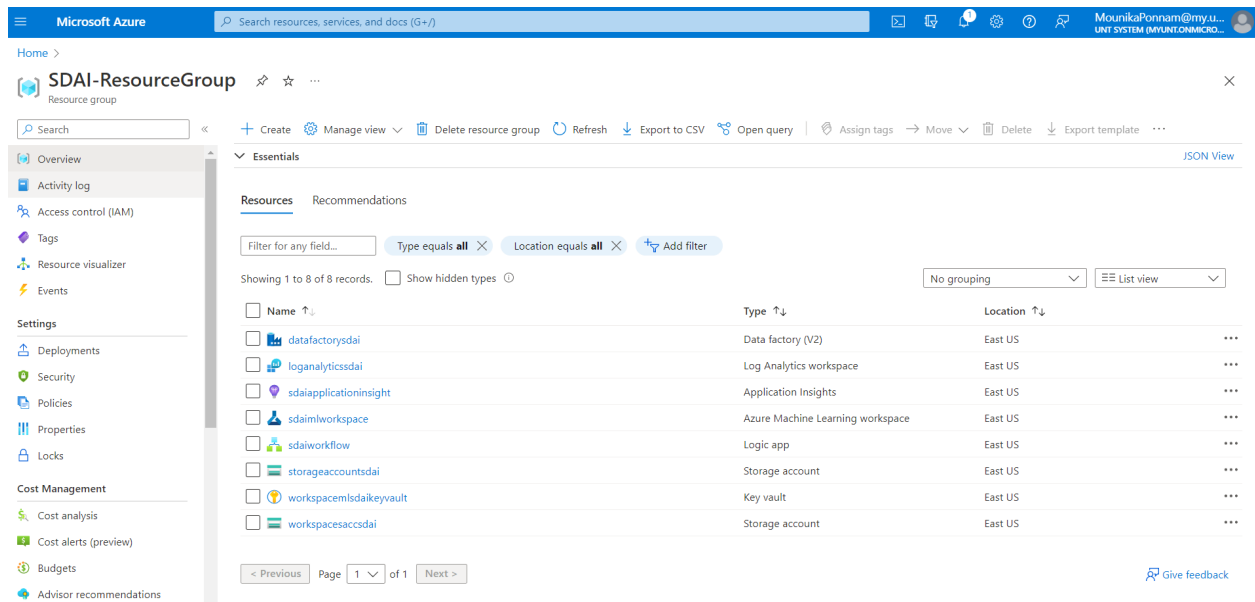
Step 8 : To actually deploy all the resources in azure platform we need to run “**terraform apply -auto-approve**” command



Step 9 : After deploying all the services in the azure portal we will get confirmation as shown in below image



After implementing all the above steps using terraform, all the azure services get deployed within minutes of time. This saves lots of time and reduces manpower as no human intervention is required, with just a few terraform commands, multiple services will be deployed. If we provision these services manually, it takes a minimum 2-3 hours. Terraform deploys multiple services simultaneously. This makes terraform very unique. On top of that, it is an open source tool and very cost efficient for the users.



In the azure portal we can see all the services deployed using terraform. This is the Auto Deployment feature in the AutoOps project. To collect all the anomaly logs that occur in the azure data platform, we need to enable Diagnostics Settings for each individual service. By doing that, It will collect all the detailed error messages at some generic repository called “Log Analytics Workspace”. After receiving all the logs in the log analytics workspace, we can filter out logs as per our requirement with respect to individual service.

Anomaly Collection Feature :

For instance, to collect the logs from Azure data factory, we will enable the diagnostic settings for this service. While configuring this feature we need to provide a destination as some storage account or log analytics workspace to store logs. In our scenario we are selecting our storage destination as Log Analytics Workspace. Then we need to select what all the logs we need to collect corresponding to the checkbox.

Microsoft Azure portal showing the overview of the **datafactorysdai** Data factory (V2) resource. The left sidebar lists navigation options, with **Diagnostic settings** highlighted in a red box. The main content area displays the resource details and the **Azure Data Factory Studio** launch button.

Essentials

- Resource group (move): [SDAI-ResourceGroup](#)
- Type: Data factory (V2)
- Status: Succeeded
- Location: East US
- Subscription (move): [Azure for Students](#)
- Subscription ID: ea340f97-de15-4ba9-b75b-0dfdbb02cdd3

Azure Data Factory Studio

[Launch studio](#)

Quick Starts **Tutorials** **Template Gallery** **Training Modules**

Monitoring

Diagnostic settings

Microsoft Azure portal showing the **Diagnostic settings** page for the **datafactorysdai** Data factory (V2) resource. The left sidebar lists navigation options, with **Diagnostic settings** highlighted in a red box. The main content area displays the diagnostic settings configuration options.

Diagnostic settings

Diagnostic settings are used to configure streaming export of platform logs and metrics for a resource to the destination of your choice. You may create up to five different diagnostic settings to send different logs and metrics to independent destinations. [Learn more about diagnostic settings](#)

Name	Storage account	Event hub	Log Analytics workspace	Partner solution	Edit setting
No diagnostic settings defined					
+ Add diagnostic setting					

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- Pipeline activity runs log
- Pipeline runs log
- Trigger runs log
- Sandbox Pipeline runs log
- Sandbox Activity runs log
- SSIS package event messages
- SSIS package executable statistics
- SSIS package event message context
- SSIS package execution component phases
- SSIS package execution data statistics
- SSIS integration runtime logs
- AllMetrics

Diagnostic setting ...

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name * Data factory Logs ✓

Logs

Category groups ①

☐ allLogs

Categories

☒ Pipeline activity runs log

☒ Pipeline runs log

☒ Trigger runs log

☐ Sandbox Pipeline runs log

Destination details

☒ Send to Log Analytics workspace

Subscription

Azure for Students

Log Analytics workspace

loganalyticsdai (eastus)

Destination table ①

Azure diagnostics Resource specific

☐ Archive to a storage account

Monitoring Feature :

After enabling the diagnostic settings, we can monitor the Azure data factory service. We can track all activities that happen inside the service like capturing the pipeline fails, trigger fails. As soon as a pipeline fails in a data factory, the information regarding anomaly and the root cause of the pipeline fail, all these details can be captured into Log analytics Workspace. In Log Analytics Workspace we can use Kusto Query language to filter out logs as per our need. In the below image I am filtering out logs based on the ErrorMessage column, if the column is not empty, that means an anomaly is detected in Azure data factory logs.

New Query 1* x + Feedback Queries

loganalyticsdai Select scope Run Time range : Last 24 hours Save Share + New alert rule Export Pin to ...

```
1 ADFActivityRun
2 | where ErrorMessage != ''
3
4
```

Results Chart

TimeGenerated [UTC]	ResourceId	OperationName	Category	CorrelationId	Level
PipelineRunId	63d010c6-8dcb-482d-90a2-233d96870140				
EffectiveIntegrationRuntime	AutoResolveIntegrationRuntime (East US)				
ActivityType	Copy				
ActivityIterationCount	1				
End [UTC]	2022-11-18T05:25:47.7722108Z				
FailureType	UserError				
PipelineName	pipeline				
> Input	{"source":{"type":"DelimitedTextSource","storeSettings":{"type":"AzureBlobStorageReadSettings","recursive":true,"wildcardFileName":"","enablePartitionDiscovery":false}}				
> Output	{"copyDuration":3,"errors":[],"effectiveIntegrationRuntime":"AutoResolveIntegrationRuntime (East US)","usedDataIntegrationUnits":4,"usedParallelCopies":1,"executionD				
ErrorCode	ActionTimedOut				
ErrorMessage	Activity timed out				

Schema and Filter

AI/ML Engine :

We have developed a model which takes error messages as input and gives us proper resolution steps to resolve the issues that occur in Azure data platform. In Order to automate the process from collecting logs to getting corresponding resolution steps, we have to publish the model inside the Azure Machine Learning Workspace. These automation activities can be put inside a logic app which will consolidate and automate all tasks from A-Z like, from collecting logs to providing resolution steps.

7. Project Management

7.1 Work completed:

- **Deployment :** People no longer have to wait days or weeks for the provisioning of Azure data services, which is the elegance of the AutoOps initiative. With the help of Terraform, we can prepare our platform in a matter of hours without any human involvement. The project is fairly unique as a result of this.
- **Monitoring:** For each and every service in the azure we can configure diagnostic settings. By doing that It will collect all logs of the services at some generic repository which we call it as Azure log analytics workspace. This helps the developer to have a look at the anomaly details in a very easy way and it helps them to focus more on the functionality.
- **Anomaly collection :** This has the additional benefit of removing the need for developers to use any third-party programs in order to save error information from the Azure data platform, which is advantageous for the AutoOps project. To gather all log and metric data on the Azure data platform, we can make use of the "Log analytics Workspace" native Azure service. The cost of this service is likewise fair.

7.2 MEMBER CONTRIBUTION TABLE:

Name	Student Id	Responsibility	Contribution
Mounika Ponnamm	11610822	Written Terraform code for azure services, Implemented Deployment, Monitoring, Anomaly detection features, Prepared document.	40
Sai Sarat Chandra Vytla	11588541	Designing Terraform scripts, Designing architectures, Helped with document preparation.	30
Rahul Manikonda	11608670	Code deployment, Helped with document preparation and diagrams.	30

7.3 Work to be completed:

- AI/AM Engine : To train the model, we use all of the common error messages and their accompanying fixes. As soon as our model's accuracy increases, we'll submit it to the Azure ML workspace. This will give us the measures we need to do in order to resolve the issue with the Azure data platform.
- Resolution : If the problem is widespread, we can either use automatic resolution, which will take care of the problems on its own, or we can create some kind of document to make it simpler for the developer to tackle the problem after receiving the resolution instructions from the AI/ML engine.

7.4 MEMBER CONTRIBUTION TABLE:

Name	Student Id	Responsibility	Issues/Concerns
Mounika Ponnamm	11610822	Developing AI/ML engine, Publishing code into Azure Machine Learning workspace	None
Sai Sarat Chandra Vytla	11588541	Building logic app workflow to automate the entire process of AutoOps	None
Rahul Manikonda	11608670	Improving machine learning model	None

8. References/Bibliography :

8.1 Azure services provisioning :

[Quickstart - Create an Azure resource group using Terraform | Microsoft Learn](#)

[Configure diagnostic settings and a workspace - Azure Data Factory | Microsoft Learn](#)

8.2 Monitoring Platform :

[Azure Monitor - Modern Observability Tools | Microsoft Azure](#)

[What is AIOps? | IBM](#)

[Advancing anomaly detection with AIOps—introducing AiDice | Azure Blog and Updates | Microsoft Azure](#)

[What Is AIOps? Artificial Intelligence for IT Operations - Cisco](#)