CMAP

Azure Cloud Deployment (PROD)

**v1.0 as of 07.03.2025**

**Document Revision History**

Each time this document is modified, increment the version number appropriately and add a new row to the table below. In the Comments column, document the changes that were made in sufficient detail.

| **Date** | **Author** | **Version** | **Comments** |
| --- | --- | --- | --- |
| 07/03/2025 | Syneren Technologies Corp | 1.0 | Created |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[1. Introduction 4](#_Toc202969920)

[2. Deliverables 4](#_Toc202969921)

[3. Cloud Services Overview 4](#_Toc202969922)

[4. Frontend Deployment 5](#_Toc202969923)

[Using Azure Static Web App 5](#_Toc202969924)

[5. Backend Deployment 7](#_Toc202969925)

[5a. Node.js Backend Using Azure App Service 7](#_Toc202969926)

[Option A: Via Portal 7](#_Toc202969927)

[Option B: Via Azure CLI 10](#_Toc202969928)

[5b. Python Backend Using Azure App Service and Azure Container Registry (ACR) 12](#_Toc202969929)

[Option A: Via Portal 12](#_Toc202969930)

[Option B: Via Azure CLI 15](#_Toc202969931)

[6. Database Setup: 16](#_Toc202969932)

[Using Azure Cosmos DB with MongoDB API 16](#_Toc202969933)

[Creating the Database 17](#_Toc202969934)

[Importing the Database 17](#_Toc202969935)

[7. Document Storage: 19](#_Toc202969936)

[Using Azure Blob Storage 19](#_Toc202969937)

[8. Logging: 20](#_Toc202969938)

[Using Azure Application Insights 20](#_Toc202969939)

[9. Key Management: 21](#_Toc202969940)

[Using Azure Key Vault 21](#_Toc202969941)

[10. Authentication 22](#_Toc202969942)

[Using Azure App Registrations 22](#_Toc202969943)

[11. Code Enviroment Changes 23](#_Toc202969944)

[10a. Frontend Environment 23](#_Toc202969945)

[10b. Backend Environment 24](#_Toc202969946)

[12. Final Notes and Best Practices: 25](#_Toc202969947)

# Introduction

This document covers deployment of CMAP frontend and backend on Azure with Deployment packages, Cosmos DB database, Blob Storage for documents, Application Insights for logging, and Key Vault for secrets management.

# Deliverables

| **Component** | **Provided As** | **Notes** |
| --- | --- | --- |
| Frontend | dist.zip | Pre-built Vite static assets |
| Backend (Node.js) | node\_backend.zip | Includes package.json, excludes node\_modules |
| Backend (Python) | python\_backend.tar (Docker image) | No local setup needed |
| Database | db.zip | MongoDB collections as JSON/BSON |

# Cloud Services Overview

| **Component** | **Technology/Service** | **Notes** |
| --- | --- | --- |
| Frontend | Azure Static Web Apps | Hosts Vite static content |
| Backend (Node.js) | Azure App Service (Linux) | ZIP deployment, auto-installs dependencies |
| Backend (Python) | Azure App Service (Linux) + Azure Container Registry (ACR) | CI/CD with Docker build & deploy pipeline |
| Database | Azure Cosmos DB (MongoDB API) | Connection string managed as env variable |
| Document Storage | Azure Blob Storage | For storing application documents |
| Logging | Azure Application Insights | Capture custom logs, traces, and telemetry |
| Key Management | Azure Key Vault | Secure storage of keys, secrets, and connection strings |
| Authentication | App Registrations | Secured Entra ID login |
| Code Environment Changes | None | Changing the URI and other variables in the Code environment |

# Frontend Deployment

## Using Azure Static Web App

Azure Static Web Apps provide a simple way to deploy static frontend apps with built-in CI/CD capabilities tightly integrated with Azure Repos.

**Steps:**

1. **Create an Azure Static Web App**
   * In Azure Portal, click **Create a resource** → Search **Static Web Apps**
   * Click **Create**
2. **Configure Static Web App**
   * **Subscription & Resource Group**: Choose your subscription and resource group
   * **Name**: Provide a name for the Static Web App
   * **Region**: Choose a region close to your users
   * **Environment**: Select **Custom**
   * **Source**: Other (for manual upload)
3. **Deployment Configuration**
   * Select ‘Deployment authorization policy’ as ‘Deployment Token’
4. **Plan and Pricing**
   * Choose an appropriate plan (Free, Standard, etc.) based on your requirements
5. **Review & Create**
   * Click **Review + Create**
   * Once validated, click **Create**
6. **Deployment Token**
   * Once the Static Web app is successfully created, go to the ‘Overview’.
   * Click on Manage Deployment Token and copy the token.
7. **Upload the code**

* Open a Command Prompt
* Make sure node is installed to use npm.
* Install Azure Static Web Apps CLI using:

npm install -g @azure/static-web-apps-cli

* If you want to get token from Azure CLI:

az staticwebapp secrets list \

--name <static-app-name> \

--resource-group <resource-group> \

--query properties.apiKey \

-o tsv

* Enter this command:

swa deploy \

--deployment-token <token>\

--app-location . \

--output-location dist

--environment production

1. **Custom Domain**

**Secure with Custom Domain and HTTPS**

* In Static Web App settings → **Custom Domains**
* Map your domain and enable HTTPS with Azure-managed certificate.

# Backend Deployment

## 5a. Node.js Backend Using Azure App Service

Deploy Node.js backend app with automated CI/CD from Azure Repos.

**Steps:**

### Option A: Via Portal

1. **Create Azure App Service**
   * In Azure Portal → **Create a resource** → Search **App Service**
   * Select **Create**
   * Configure:
     + Subscription and Resource Group
     + Name for the App Service
     + Publish: **Code**
     + Runtime stack: **Node.js LTS version** (choose latest stable LTS)
     + Operating System: **Linux**
     + Region: Select region close to users
     + Hit Create
   * Deployment using UI

(Sometimes this ‘Zip Push UI’ is not available, depending on region and service type)

* + - Go to **Development Center → Zip Deploy** OR use the Kudu **Tools → Zip Push Deploy**
    - Upload the zipped node\_backend.zip file
    - If Zip Push Deploy is not available, please use the below methods.
  + Deployment using FTPS/CURL/BASH/SSH
    - Go to**Deployment → Deployment Center →** Click on **FTPS Credentials**
    - Note down the **FTPS endpoint** and FTPS username and password if available
    - If the FTPS endpoint is not available, then go to **Settings → Configuration →** Turn on **FTP Basic Auth Publishing Credentials**
    - Hit **Save**
    - Use the following ‘curl’ (Windows/Linux) or ‘lftp’ (Linux/Mac) command to upload the backend.zip file to the ‘App Service:

Windows/Linux ->

curl -T backend.zip <ftps endpoint>/backend.zip --user <username>:<password> --ssl-reqd

(NOTE: use –user only if username and password are available)

Linux/Mac ->

lftp -u <username>,<password> ftps://<ftps-host>

# Then inside lftp shell:

cd /site/wwwroot

put backend.zip

bye

* + - Open

<https://cmap-be-prod.scm.azurewebsites.net/wwwroot/>

OR

* + - * Go to **Development Tools** → **Advanced Tools** → **Go**
      * In Browse Directory, click on ‘Site wwwroot’
      * Check if your uploaded backend.zip is there or not
    - Now that the zip file is uploaded to the wwwroot, you can use SSH/BASH to unzip and deploy it.
    - For WebSSH:

https://<your-app-name>.scm.azurewebsites.net/webssh/host

OR

* + - * Go to **Development Tools** → **SSH**

OR

* + - * Go to **Development Tools** → **Advanced Tools** → **Go**
      * Click on SSH from the Top navigation bar/ribbon
    - For BASH:

https://<your-app-name>.scm.azurewebsites.net/DebugConsole

OR

* + - * Go to **Development Tools** → **Advanced Tools** → **Go**
      * Click on BASH from the Top navigation bar/ribbon
    - For Local SSH:

ssh <username>@<your-app-service-name>.scm.azurewebsites.net

The username is usually root or your app service user

* + - After using any of the above methods to connect, now comes the unzipping, installing and deploying
    - Unzip:

unzip backend.zip -d .

* + - Remove backend.zip and starter file:

rm backend.zip

rm hoststart.html (or similar name, use ‘ls’ or browse wwwroot to confirm)

rm -rf backend (for folder if needed)

* + - Install packages:

npm install

* + - No need to run using ‘npm start’, as the app will restart and we will set auto-build to be true in the configuration but if the app doesn’t restart then do ‘npm start’
  + Restrict Networking
    - Navigate to **Networking** → **Access Restrictions**
    - Add an **Allow** rule only for your Static Web App’s **outbound IPs**
    - Add a **Deny** rule for 0.0.0.0/0 to block all other public access if not needed

1. **Configure App Settings**
   * Go to your App Service → **Settings** → **Environment Variables**
   * Add necessary environment variables such as:
     + MONGO\_URI (your MongoDB connection string)
     + SCM\_DO\_BUILD\_DURING\_DEPLOYMENT (true)
     + Any other environment-specific variables
   * Go to your App Service → **Configuration**
     + Set the ‘Startup Command’ as ‘npm start’
2. **Enable Logging**
   * Go to **App Service Logs**
   * Enable **Application Logging (Filesystem)**
   * Set retention as needed for troubleshooting
3. **Restart the App Service after applying settings**

### Option B: Via Azure CLI

1. **Create App Service Plan and App (Linux, Node.js)**

az group create --name <resource-group> --location eastus

az appservice plan create --name cmap-plan --resource-group <resource-group> --sku B1 --is-linux

az webapp create --resource-group <resource-group> --plan cmap-plan --name <app-name> --runtime "NODE|22-lts"

1. **Deploy using ZIP and auto-install dependencies**

az webapp deploy \

--resource-group <resource-group> \

--name <your-app-service-name> \

--src-path backend.zip \

--type zip

1. **Set environment variables**

az webapp config appsettings set --name <app-name> --resource-group <resource-group> --settings \

MONGO\_URI="<value>" \

APPINSIGHTS\_INSTRUMENTATIONKEY="<key>" \

STORAGE\_CONNECTION\_STRING="<value>"

SCM\_DO\_BUILD\_DURING\_DEPLOYMENT=true

1. **Apply Access Restrictions**

az webapp config access-restriction add \

--resource-group <rg> \

--name <app-name> \

--rule-name AllowStaticWebApp \

--priority 100 \

--action Allow \

--ip-address <static-webapp-ip>

1. **Restart the app service**

az webapp restart --name <app-service-name> --resource-group <resource-group>

**Result:**

* Backend app runs on Azure App Service with configured environment variables.

**Helpful commands:**

* To show the current Node version

az webapp show --resource-group <resource-group-name> --name <app-name> --query siteConfig.linuxFxVersion -o tsv

* To set the current node version   
  az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "NODE|22-lts"

## 5b. Python Backend Using Azure App Service and Azure Container Registry (ACR)

For Python backend, containerize your app and deploy on Azure App Service with Docker containers, using Azure Container Registry (ACR) to store images.

**Steps:**

### Option A: Via Portal

* 1. **Create a Storage Account**
* Go to the [Azure Portal](https://portal.azure.com)
* Search for **Storage accounts** in the top search bar
* Click **+ Create**
* Fill in the form:
  + **Subscription**: your Azure subscription
  + **Resource group**: select or create (e.g., cmap-rg)
  + **Storage account name**: e.g., cmapstorage (must be globally unique)
  + **Region**: e.g., East US
  + **Performance**: Standard
  + **Redundancy**: LRS (default is fine)
* Click **Review + Create** → then **Create**
  1. **Create a Container in the Storage Account**
* Once deployment completes, click **Go to resource**
* In the left menu, under **Data storage**, click **Containers**
* Click **+ Container**
* Name it: docker-images
* Set **Public access level** to Private (no anonymous access)
* Click **Create**
  1. **Upload the .tar File to the Container**
* Inside the docker-images container, click **Upload**
* In the **Upload blob** panel:
* Click **Browse for files** and select cmap-backend.tar
* Leave blob type as Block blob
* Click **Upload**
  1. **Generate a SAS Token for the .tar Blob**
* In the list of blobs, find cmap-backend.tar
* Click the ... (three-dot menu) or right-click the blob → **Generate SAS**
* In the SAS settings:
  + **Permissions**: Check **Read**
  + **Expiry**: Set 24 hours (or desired duration)
* Click **Generate SAS token and URL**
* Copy the **Blob SAS URL** (looks like https://...tar?...) — this is used in the next step
  1. **Create an Azure Container Registry (ACR)**
* In the Azure Portal, search for **Container registries**
* Click **+ Create**
* Fill in the form:
  + **Subscription** and **Resource Group**
  + **Registry Name**: cmapacr (must be globally unique)
  + **Location**: same as App Service / Storage (e.g., East US)
  + **SKU**: Basic
* Click **Review + Create** → then **Create**
  1. **Import Docker Image into ACR from Blob URL**
* After ACR is deployed, go to your ACR (cmapacr)
* In the left menu, click **Repositories**
* Click **+ Import**
* In the **Import Image** form:
  + **Source type**: *Other registry*
  + **Source image**: paste the **Blob SAS URL**
  + **Repository name**: cmap-backend
  + **Tag**: prod
  + Leave **Username/Password** empty
  + Enable **Force import** if re-importing
* Click **Review + Import**
  1. **Create Azure App Service (Linux, Container)**
* Create new App Service
* In Publish, choose Docker Container
* Set Linux OS
* Configure container settings to pull from your ACR image
* Image Source: Azure Container Registry
* Registry: select your ACR (e.g., cmapacr)
* Image: select your image (e.g., cmap-backend)
* Tag: select your tag (e.g., prod)
* Restrict Networking
  + Navigate to **Networking** → **Access Restrictions**
  + Add an **Allow** rule only for your Static Web App’s **outbound IPs**
  + Add a **Deny** rule for 0.0.0.0/0 to block all other public access if not needed
  1. **Configure App Settings**
* Add environment variables (e.g., MONGO\_URI) in the App Service configuration
  1. **Enable Logging**
* Enable container logs and App Service logs as needed

### Option B: Via Azure CLI

* 1. **Create a Storage Account (if not already)**

az storage account create \

--name cmapstorage \

--resource-group cmap-rg \

--location eastus \

--sku Standard\_LRS

* 1. **Create a Container in the Storage Account**

az storage container create \

--name docker-images \

--account-name cmapstorage \

--public-access off

* 1. **Upload the .tar File to the Container**

az storage blob upload \

--account-name cmapstorage \

--container-name docker-images \

--name cmap-backend.tar \

--file cmap-backend.tar

* 1. **Generate a SAS Token for the Blob (Temporary URL)**

az storage blob generate-sas \

--account-name cmapstorage \

--container-name docker-images \

--name cmap-backend.tar \

--permissions r \

--expiry $(date -u -d "1 day" '+%Y-%m-%dT%H:%MZ') \

--output tsv

* 1. **Create ACR (if not already):**

az acr create --name cmapacr --resource-group cmap-rg --sku Basic

* 1. **Import image into ACR from Blob SAS URL:**

az acr import \

--name cmapacr \

--source "<sas-url-to-cmap-backend.tar>" \

--image cmap-backend:prod \

--registry cmapacr \

--force

* 1. **Create App Service (Linux, Container)**

az appservice plan create \

--name cmap-plan \

--resource-group cmap-rg \

--sku B1 --is-linux

az webapp create \

--resource-group cmap-rg \

--plan cmap-plan \

--name cmap-backend-app \

--deployment-container-image-name cmapacr.azurecr.io/cmap-backend:prod

* 1. **Set environment variables**

az webapp config appsettings set \

--name cmap-backend-app \

--resource-group cmap-rg \

--settings \

MONGO\_URI="<value>" \

APPINSIGHTS\_INSTRUMENTATIONKEY="<value>"

* 1. **Apply Access Restrictions**

az webapp config access-restriction add \

--resource-group <rg> \

--name cmap-backend-app \

--rule-name AllowStaticWebApp \

--priority 100 \

--action Allow \

--ip-address <static-webapp-ip>

# Database Setup:

## Using Azure Cosmos DB with MongoDB API

Azure Cosmos DB offers a fully managed, scalable NoSQL database service. For MongoDB-compatible apps, Cosmos DB provides a MongoDB API.

### Creating the Database

**Steps:**

1. **Create Cosmos DB Account**
   * Go to Azure Portal → **Create a resource** → Search **Azure Cosmos DB**
   * Click **Create**
   * Select **API** as **MongoDB**
   * Choose the subscription and resource group
   * Provide an account name
   * Choose region closest to your backend App Service for low latency
   * Select capacity mode and other settings as needed
   * ***Security:*** Enable **Data Encryption** (enabled by default with Microsoft-managed keys, optionally bring your own key [CMK])
2. **Secure Networking (Highly Recommended)**
   * Before proceeding further:
   * Go to your Cosmos DB resource → **Networking**
   * Set **Public network access** to:
     + Disabled (to block external access)
   * Enable **Private Endpoint** (optional, for internal Azure access only)
   * Add **Firewall rules** (if public access is enabled temporarily), specifying:
     + Your App Service outbound IPs only

This ensures that **only trusted services** (e.g., your backend App Service) can access the database.

1. **Get Connection String**
   * Once the Cosmos DB account is created, navigate to **Keys**
   * Copy the **Primary Connection String**
2. **Use Connection String**
   * Set the copied connection string as the MONGO\_URI environment variable in your backend App Service configuration (Node.js)
   * Example:

mongodb://username:password@clustername:port/dbname?ssl=true&replicaSet=globaldb&retrywrites=false&maxIdleTimeMS=120000&appName=@your-account@

### Importing the Database

#### Option A: Via MongoDB Compass (For Initial Setup)

**Steps:**

1. Install MongoDB Compass
   * Download MongoDB Compass for your operating system
2. Unzip DB file
   * Unzip the db.zip, and extract the ‘cmap’ folder
3. Create the connection with the Cosmos MongoDB URI
   * Click on New Connection
   * Enter the MongoDB URI (something like this:

mongodb+srv://<username>:<password\_with\_%for\_special\_char>@<cluster-name>.global.mongocluster.cosmos.azure.com/<dbname>?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000

* + Click on Save and Connect

1. Create new DB
   * Click on New Database
   * Enter the Database name as ‘cmap’
   * Enter the collection name as ‘user’
2. Import User
   * Once the user collection is created, click on import and choose the user.json from cmap extracted db
3. Import Systems
   * Once the systems collection is created, click on import and choose the systems.json from cmap extracted db
4. Continue or Exit
   * You can continue doing same for all collections if you need.
   * Download MongoDB Tools to use this command to import the whole new db if it works:

mongorestore --db cmap your\_extracted\_db\_location\cmap

#### Option B: Via CMAP Web Application

**Steps:**

Once the user and systems collection are added to the db you can continue adding others via CMAP Web Application UI.

1. Login to CMAP via Global Admin
2. Click on Import/Export DB button
   * Click on Import tab
   * Choose the db.zip file
   * Select Append/Override as per your choice
   * Keep the db name as cmap
   * Hit Import

# Document Storage:

## Using Azure Blob Storage

Azure Blob Storage is used to store unstructured data like documents, images, or any files your application needs to serve or process.

**Steps to set up:**

1. **Create an Azure Storage Account**
   * In Azure Portal → **Create a resource** → Search **Storage account**
   * Choose subscription, resource group, name, and region (preferably same region as your app)
   * Choose **Standard** performance and **BlobStorage** or **General Purpose v2** account kind
2. **Create Blob Container**
   * In the Storage Account → **Containers** → **+ Container**
   * Name the container (e.g., documents)
   * Set Public access level as needed (usually private)
3. **Get Access Credentials**
   * Use either:
     + **Connection String** (found under Storage Account → Access keys)
     + Or better: Use **Azure Key Vault** (covered below) to securely store these keys
4. **Integrate with Application**
   * Use Azure SDKs (e.g., @azure/storage-blob for Node.js, azure-storage-blob for Python) to upload/download documents programmatically
   * Store the connection string or SAS tokens in environment variables or Azure Key Vault

# Logging:

## Using Azure Application Insights

Azure Application Insights provides deep application performance monitoring, telemetry, and the ability to log custom events and traces.

**Setup and Integration:**

1. **Create Application Insights Resource**
   * In Azure Portal → **Create a resource** → Search **Application Insights**
   * Choose subscription, resource group, region, and application type (e.g., Node.js or Python)
2. **Configure Backend Application**
   * **Node.js:**
     + Install Application Insights SDK:

npm install applicationinsights

* + - Initialize in your code (e.g., app.js):

Examle: Track custom event

client.trackEvent({ name: "custom event", properties: { customProperty: "value" } });

* + **Python:**
    - Install Application Insights SDK:

1. **Set Environment Variable**
   * Add environment variable APPINSIGHTS\_INSTRUMENTATIONKEY to your App Service (or container environment) with the Instrumentation Key from the Application Insights resource.
2. **Monitor and Query Logs**
   * Use the Azure Portal Application Insights blade to:
     + View performance metrics
     + Analyze request rates and failures
     + Query custom events and traces using Kusto Query Language (KQL)
     + Set up alerts and dashboards for proactive monitoring

# Key Management:

## Using Azure Key Vault

Azure Key Vault securely stores secrets, keys, and certificates used by your applications.

**Steps:**

1. **Create Azure Key Vault**
   * Azure Portal → **Create a resource** → Search **Key Vault**
   * Enter name, subscription, resource group, region
2. **Add Secrets / Keys**
   * In your Key Vault → **Secrets** → **+ Generate/Import**
   * Add secrets such as:
     + Cosmos DB connection string
     + Storage account keys or SAS tokens
     + Any other sensitive credentials
3. **Access Key Vault from App Services**
   * Grant your App Service managed identity **Get** permissions on the Key Vault secrets via Access Policies
   * In your application, use Azure SDK or environment variables referencing Key Vault references (e.g., via @Microsoft.KeyVault syntax in App Settings)
4. **Use Key Vault references in App Service**
   * In App Service Configuration, set values like

@Microsoft.KeyVault(SecretUri=https://<your-vault-name>.vault.azure.net/secrets/<secret-name>/<secret-version>)

* + This allows your app to retrieve secrets securely at runtime without hardcoding them

# Authentication

## Using Azure App Registrations

Azure App Registrations allow a third-party application to interact with Microsoft APIs like msal, graph. It can also be used to login via Microsoft Entra ID.

**Steps:**

1. **Go to App Registrations**
   * Azure Portal → **App Registrations** → Click New Registration
   * Enter name
     + CMAP\_AUTH
   * Supported account types
     + Choose Accounts in this organizational directory only if you want only that org to have access to the web application else choose Accounts in any organizational directory
   * Redirect URI (optional at this step)
     + Select Single-page application and enter the Custom Domain URL of frontend
2. **Go to Authentication**
   * Add a platform
   * Select Single-page application
   * Click add uri, and enter the frontend URI
   * Hit save
3. **Go to Certificates & Secrets**
   * Click on New client secret
   * Enter Description as ‘CMAP secret’
   * Choose expiration duration
   * Hit ‘Add’
4. **Go to Expose an API**
   * Click on Add a scope
   * Put the scope name as access\_as\_user
   * Select ‘Admins and Users’ in Who can Consent?
   * Admin consent display name: ‘Access CMAP API’
   * Admin consent description: ‘Allows the app to call CMAP API on your behalf.’
   * State: Enabled
   * Hit Save
5. **Go to API Permissions**
   * Click ‘Add a permission’
   * Select Microsoft Graph
   * Choose Delegated Permissions
   * From the dropdown/search bar, select **email**, **openid**, and **profile**
   * Hit Add permissions.
   * Again click ‘Add a permission’
   * Select My APIs and choose ‘CMAP\_AUTH’
   * Choose Delegated Permissions
   * From the dropdown/search bar, select **access\_as\_user**
   * Hit Add permissions
6. **Go to Overview and Note Down the Application (Client) ID and Directory (Tenant) ID.**

# Code Enviroment Changes

## 10a. Frontend Environment

**Steps:**

**Open ‘.env’ file in frontend code root**

Replace the following values

**Authentication Changes:**

* + VITE\_AZURE\_CLIENT\_ID: (put **Application (Client) ID** that you noted in 9. Authentication Step)
  + VITE\_AUDIENCE: (put **Application (Client) ID** that you noted in 9. Authentication Step)
  + VITE\_TENANT\_ID: (put **Directory (Tenant) ID** that you noted in 9. Authentication Step)
  + VITE\_REDIRECT\_URI: (put the **frontend URL**)

**URI Changes:**

* + VITE\_API\_URL: (put the **backend API URL**)
  + PYTHON\_API\_URL: (put the **python backend API URL**)

## 10b. Backend Environment

**Steps:**

**Open ‘.env’ file in node.js backend code root**

Replace the following values

**Authentication Changes:**

* + AZURE\_CLIENT\_ID: (put **Application (Client) ID** that you noted in 9. Authentication Step)
  + AUDIENCE: (put **Application (Client) ID** that you noted in 9. Authentication Step)
  + TENANT\_ID: (put **Directory (Tenant) ID** that you noted in 9. Authentication Step)

**URI Changes:**

* + FRONTEND\_URL: (put the **frontend URL**)

**PORT Changes:**

* + PORT: (put the **PORT** in use for backend)

**DB Changes:**

MONGO\_URI: (put the **Cosmos MongoDB URI**, somethinglike this mongodb+srv://<username>:<password\_with\_%for\_special\_char>@<cluster-name>.global.mongocluster.cosmos.azure.com/<dbname>?tls=true&authMechanism=SCRAM-SHA-256&retrywrites=false&maxIdleTimeMS=120000

**Third-party Credentials Changes:**

* + OUTLOOK\_EMAIL: (put the email\_id of outlook from which you want to send emails)
  + OUTLOOK\_EMAIL\_PWD: (enter the password of that outlook account)
  + SN\_USERNAME: (put the username of ServiceNow from which you want to add support tickets)
  + SN\_PASSWORD: (enter the password of that ServiceNow account)

**Keys/Secret:**

If you don’t want to handle Keys/Secrets in Azure Key Vault, then you will have to handle them locally in the environment.

* + FILE\_ENCRYPTION\_KEY: (put the encryption key aes-256-gcm)
  + FILE\_SECRET: (put the secret aes-256-cbc)

# Final Notes and Best Practices:

* **Regional Proximity**: Ensure that all services (App Services, Cosmos DB, Blob Storage, Application Insights, and Key Vault) are created in the same region for **low latency**, **reduced costs**, and **better performance**.
* **Secure Secrets Management**: Store sensitive data like MONGO\_URI, APPINSIGHTS\_INSTRUMENTATIONKEY, and storage keys in **Azure Key Vault** instead of hardcoding them or using plain environment variables.
* **Managed Identities**: Use **Azure Managed Identity** to grant access to Key Vault and other services without requiring credentials in your code.
* **Custom Logging**: Use **Azure Application Insights** to track:
* Backend performance
* Request failures and exceptions
* Custom events and traces (e.g., user actions, debug info)
* **Document Storage**: Use **Azure Blob Storage** for storing documents, files, or media. Access securely using **SAS tokens** or credentials stored in Key Vault.
* **Monitoring and Alerts**: Configure alerts in Application Insights for:
* Failed requests
* Performance bottlenecks
* Custom error tracking  
  Use Kusto Query Language (KQL) for advanced diagnostics and dashboards.
* **Containerization (Python)**: Containerizing the Python backend via Docker and deploying through **Azure Container Registry (ACR)** + App Service ensures consistent environments and reproducible builds.
* **Logging Enablement**: Always enable **App Service diagnostics logging** alongside Application Insights for holistic observability.
* **Scalability**: Azure services like Static Web Apps, Cosmos DB, and Blob Storage scale seamlessly, supporting future growth in users and data.