

# Container Apps Provisioning using Terraform

---

- We will provision the Container App using Terraform as an Infrastructure as Code.
  - We will deploy it in a custom Virtual Network for isolation.
  - We will connect the Container App to ACR for Docker Image.
  - We will also create a Storage Account Container to store the `.env` file.
  - Also will deploy MySQL Flexible to store the relational data and connect it to the Container App.
- 

## Prerequisites

---

1. Azure Account with Subscription.
  2. Terraform installed.
- 

## Write Terraform Configuration files

---

First, we will write Terraform configuration files for Azure resources using predefined modules available on the internet.

## Steps

1. Create the **container-apps-terraform** directory.
2. The folder structure for the above-created directory is as follows:

```
container-apps-terraform
├── .terraform.lock.hcl
├── locals.tf
├── main.tf
├── outputs.tf
├── providers.tf
├── terraform.tfstate
├── terraform.tfstate.backup
└── .terraform
```

We need to only create *providers.tf*, *main.tf*, *outputs.tf*, & *locals.tf* file. Other files are generated while initiating terraform.

3. Create a *providers.tf* file inside the above-created directory.
4. Inside it, define the following:
  - terraform
    - required\_providers
  - provider
    - azurerm
5. Click [code](#) for reference.

6. The definition of *providers.tf* file is complete.
7. Now, create the *main.tf* file.
8. Inside *main.tf* file, we will use the following predefined modules:
  - module.resource-group
  - module.virtual-network
  - module.acr
  - module.mysql-flexible
  - module.storage
  - module.container-apps-setup
  - module.container-apps
9. Click [code](#) for reference.
10. The definition of *main.tf* file is complete.
11. Now we will create *outputs.tf* file.
12. Inside it, define the following outputs.
  - output.acr-login-server
  - output.acr-admin-username
  - output.acr-admin-password
  - output.DB\_HOST
  - output.container-apps-url
13. Click [code](#) for reference.
14. The definition of *outputs.tf* file is complete.
15. Now we will create *locals.tf* file.
16. Inside it, define the following variables:
  - local.resource-group-properties
  - local.virtual-network-properties
  - local.acr-properties
  - local.mysql-flexible-properties
  - local.storage-properties
  - local.container-apps-setup-properties
  - local.container-apps-properties
17. Click [code](#) for reference.
18. The definition of *locals.tf* file is complete.

Ensure you give the appropriate values to the variables defined in *locals.tf* file.

Also update the *sb-object-source-path* variable under *storage-properties* with local *.env* file relative path.

---

# Provisioning the Infrastructure

Now we will provision the Azure infrastructure by applying the above-created configuration files.

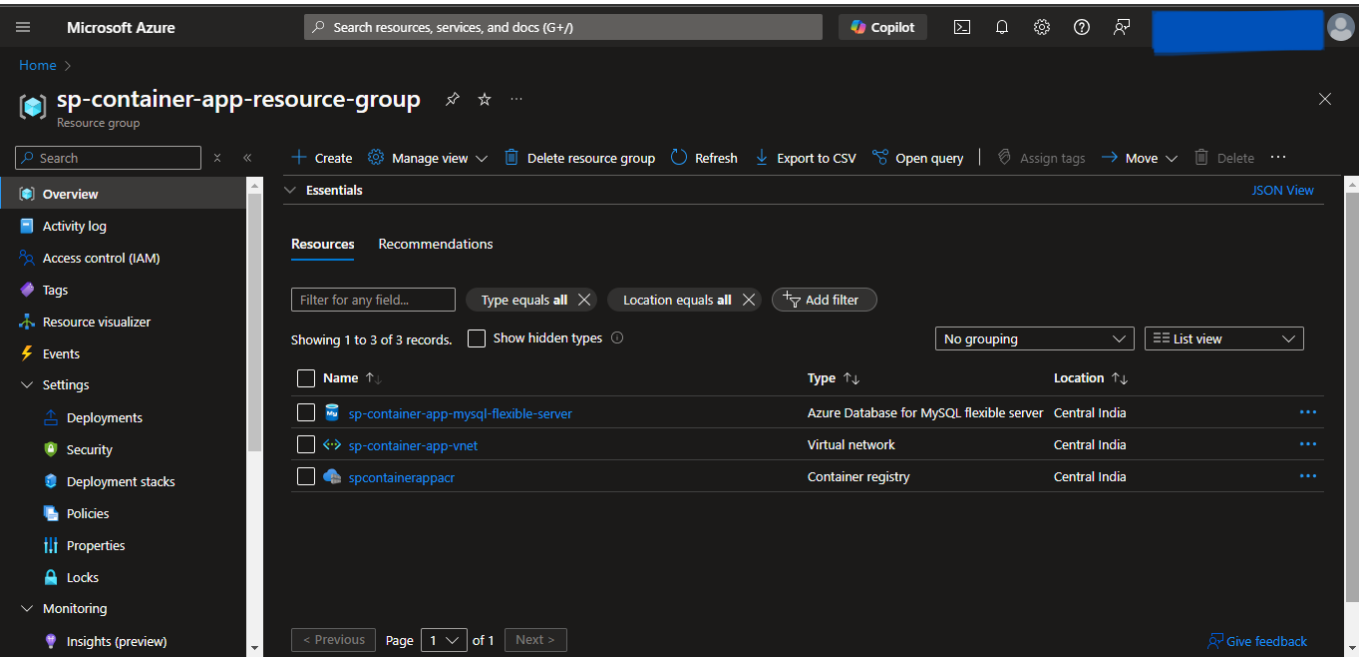
- Ensure Azure CLI is configured with appropriate Azure Account credentials and enough permissions.
- Also first provision the ACR, push the Docker Image, and then provision the Container App. To do that, comment out the **container-app** module and follow the further steps.

## Steps:

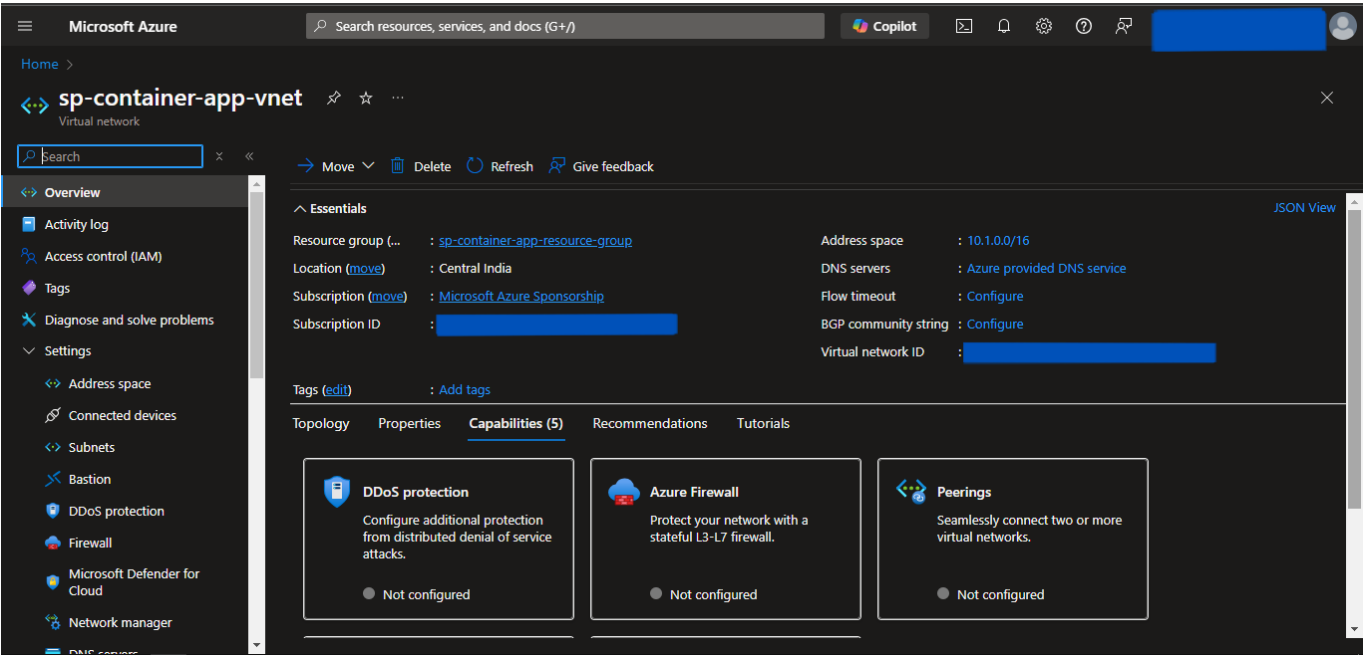
1. Open the PowerShell Window.
2. Change the directory to the above-created **container-apps-terraform** directory using the **cd** command.
3. Run the **terraform fmt -recursive** command to format the syntax of the files.
4. Run the **terraform init** command to initialize the *terraform*.
5. Run the **terraform validate** command to validate the configuration files.
6. Run the **terraform plan** command to plan the resources to be created.
7. Run the **terraform apply** command and if prompted, type **yes** to provision the infrastructure.
8. Once completed, head to the Azure Console, and verify the created resources.

# Screenshots of Provisioned Infrastructure

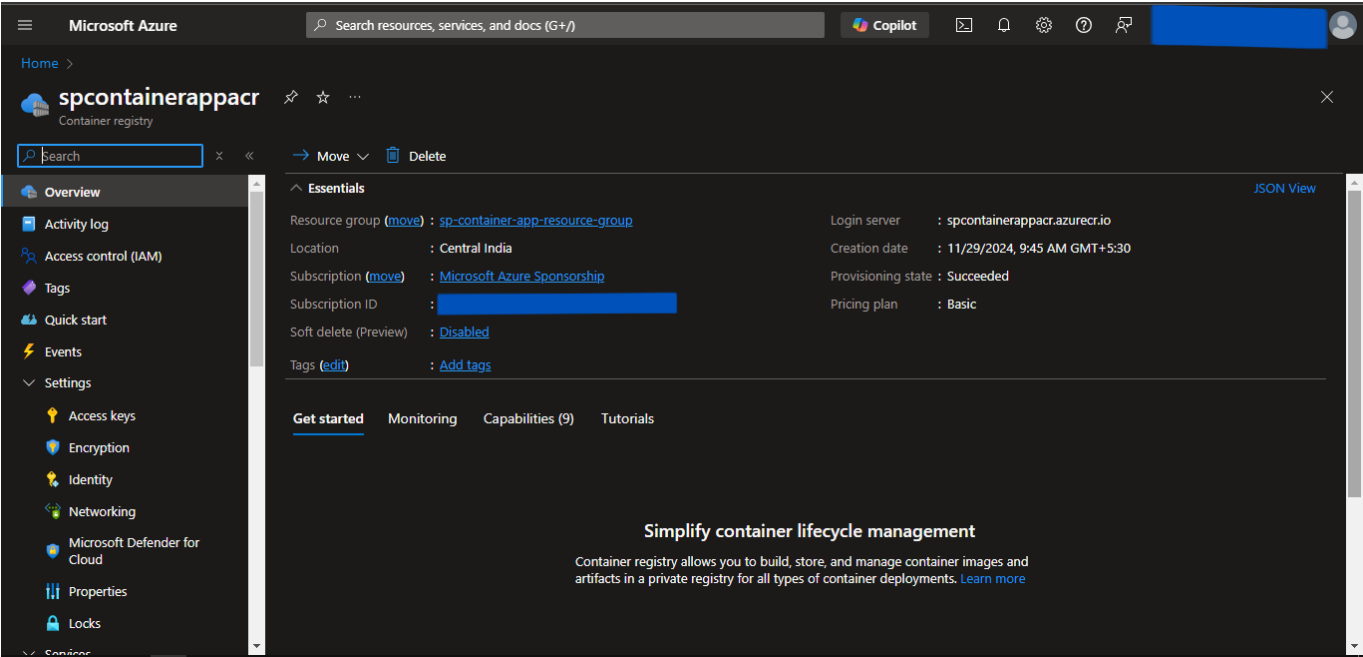
## Resource Group Image



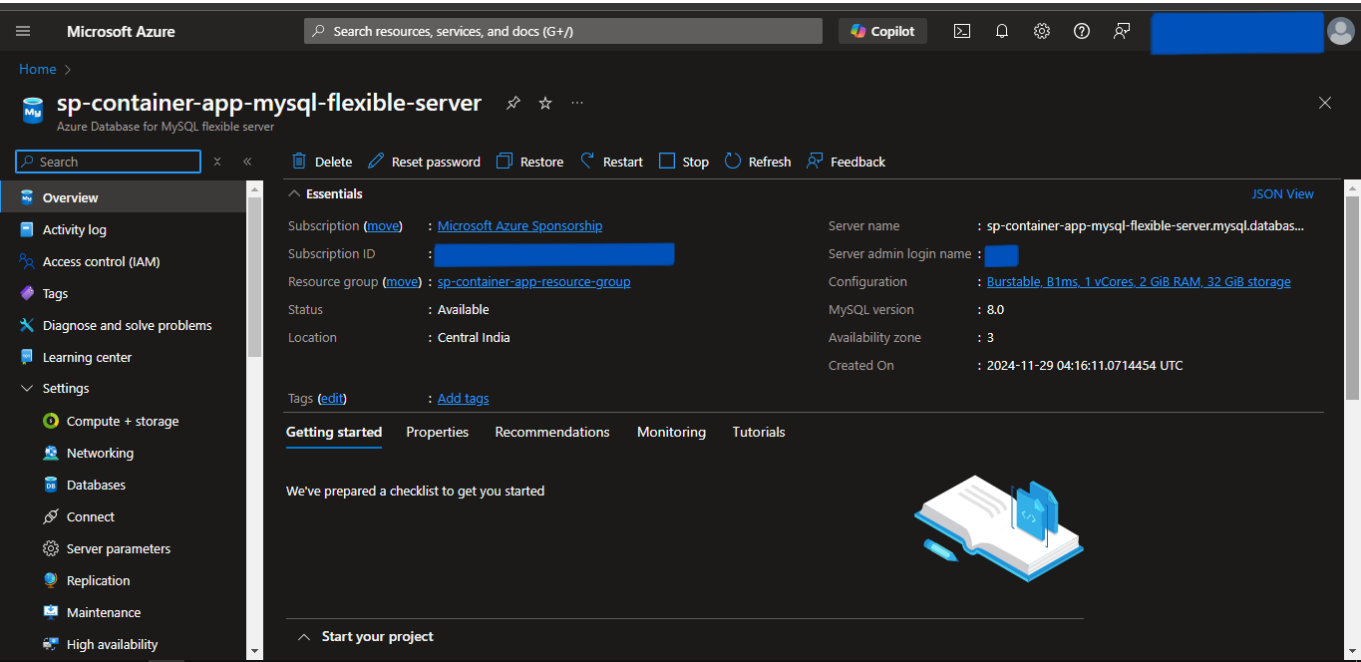
Virtual Network Image



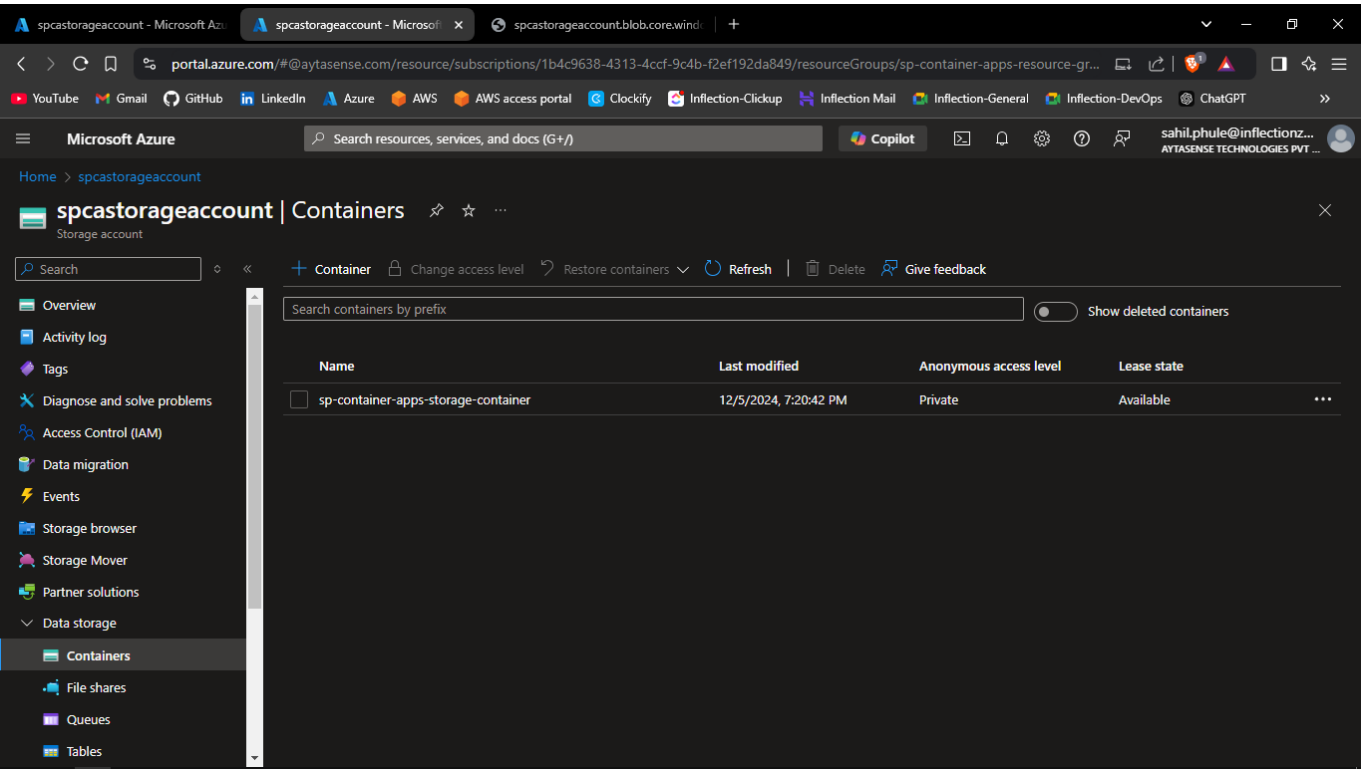
ACR Image



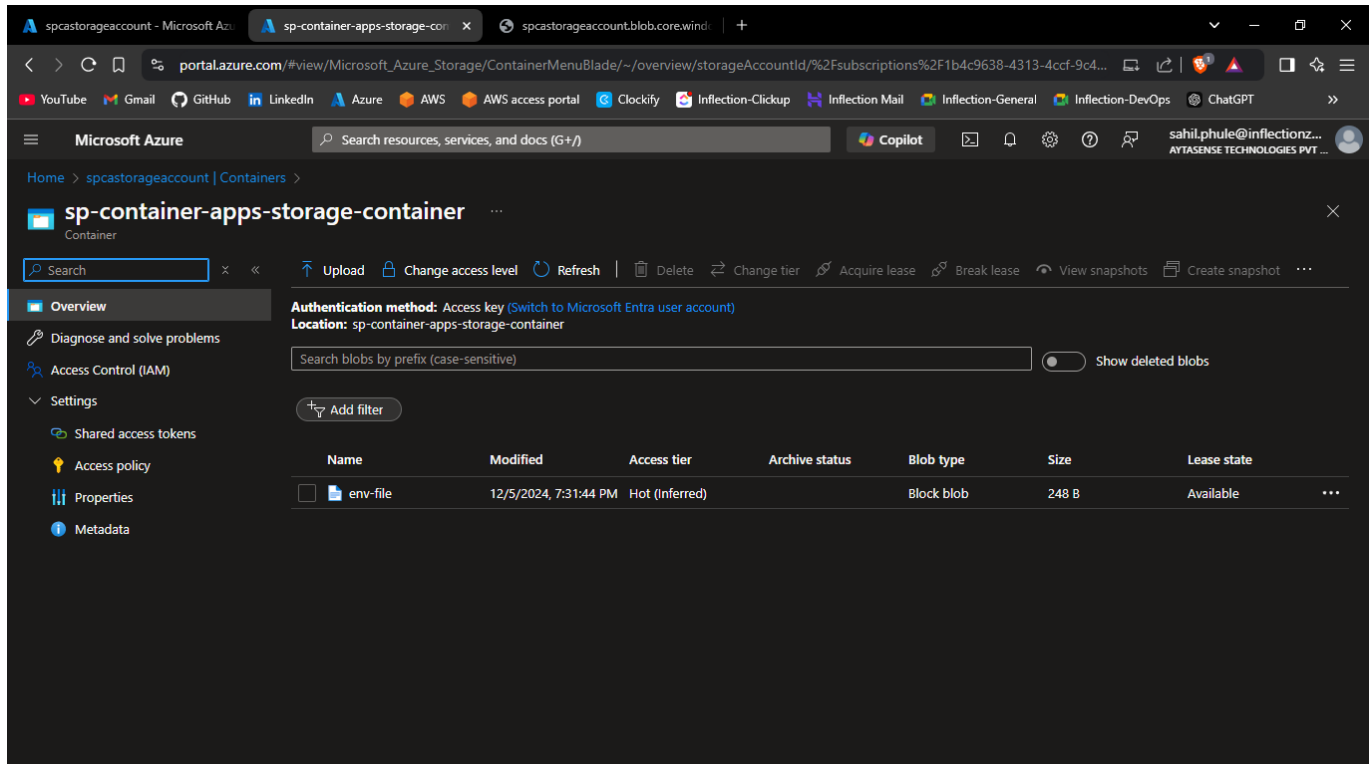
MySQL Flexible Server Image



Storage Account Container Image



## Storage Account Container Env File



## Now push the Docker Image to ACR

### Steps

1. Open a new Powershell window.
2. Run the following commands to log into ACR:

```
az login
az acr login --name <acr-name>
```

3. Then tag & push the docker image using the following commands:

```
docker tag <image-name:tag> <acr-name>.azurecr.io/<image-name:tag>
docker push <acr-name>.azurecr.io/<image-name:tag>
```

Substitute `<acr-name>` with the value defined in the above-created `locals.tf` file. Also, substitute `<image-name:tag>` with its respective name.

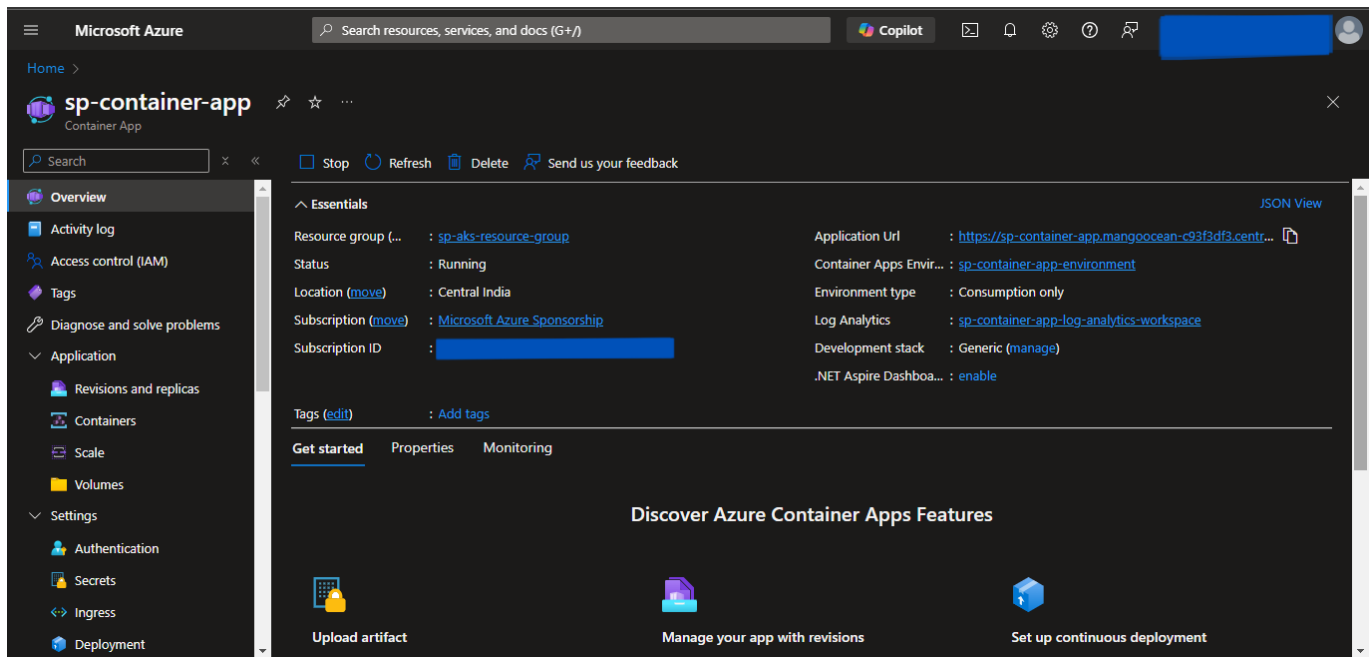
## Provisioning the Container App

Uncomment the **container-app** module that we commented on earlier and follow the further steps.

### Steps:

1. Change window to the PowerShell.
2. Run the **terraform fmt -recursive** command to format the syntax of the files.
3. Run the **terraform init** command to initialize the *terraform*.
4. Run the **terraform validate** command to validate the configuration files.
5. Run the **terraform plan** command to plan the resources to be created.
6. Run the **terraform apply** command and if prompted, type **yes** to provision the infrastructure.
7. Once completed, head to the Azure Console, and verify the created resources.
8. Run the **terraform output** command to get the values of defined variables in *outputs.tf* file.
9. Then,
  - Copy the *container-app-url*.
  - Paste the address in the browser to access the application.

## Container Apps Image



The screenshot displays the Microsoft Azure portal interface for a Container App named 'sp-container-app'. The left sidebar shows the navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Application, Revisions and replicas, Containers, Scale, Volumes, Settings, Authentication, Secrets, Ingress, and Deployment. The main content area is divided into two sections. The top section, titled 'Essentials', provides key information about the app: Resource group (sp-aks-resource-group), Status (Running), Location (Central India), Subscription (Microsoft Azure Sponsorship), and Subscription ID. It also lists the Application Url, Container Apps Environment, Environment type (Consumption only), Log Analytics, Development stack (Generic), and .NET Aspire Dashboard status. The bottom section, titled 'Discover Azure Container Apps Features', includes three cards: 'Upload artifact', 'Manage your app with revisions', and 'Set up continuous deployment'.

---

## Destroy the provisioned infrastructure

---

Lastly, we will destroy the above-created resources.

### Steps

1. To destroy infrastructure, open the Powershell Window and change the directory to the above-created **container-apps-terraform** directory using the **cd** command.
  2. Run **terraform destroy** & if prompted, type **yes**.
  3. Infrastructure will be destroyed.
-