Terraform Cookbook

Table of Contents

AWS

Azure

Deploy Svelte Website on EC2 using Terraform

- We will provision the EC2 for Static & Dynamic Web Hosting.
- We will deploy it in a custom Virtual Private Cloud for isolation.
- We will configure the server with Nginx as a Reverse Proxy and attach an SSL Certificate for secure web access.
- We will also create a Route53 Hosted Zone and add a Route53 Record to access the website using the domain name.
- We will create all these resources using Terraform as an Infrastructure as Code.

Prerequisites

1. AWS Account with IAM User Access Keys
2. Terraform installed
3. Website repository
4. Domain name

Generate SSH Key Pair

First, we will generate an **SSH Key Pair** to SSH into the EC2 server once it is created and start the Svelte with Nodejs.

Steps

1. Open the Powershell Window.
2. Run the following command to generate the ssh key pair and enter the location where you want to save the ssh key (C:/Users/user/.ssh/aws/ssh-keys/):

```
ssh-keygen -t ed25519
```

3. SSH Key will be generated.

Write Terraform Configuration files

Now we will write Terraform configuration files for AWS resources using predefined modules available on the internet.

Steps

1. Create the **ec2-website-terraform** project directory.

2. The folder structure for the above-created directory is as follows:

```
ec2-website-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
        - aws
5. Click code for reference.
6. The definition of *providers.tf* file is complete.
7. Now create the *main.tf* file.
8. Define the following modules inside it:
    - module.vpc
    - module. ec2
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.ec2-instance-public-ip
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.aws-region
    - local.vpc-properties
    - local.ec2-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.
> In *ec2-properties.ec2-instance-public-key*, provide the above generated public SSH key path.

Provisioning the Infrastructure

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

> Ensure AWS CLI is configured with appropriate IAM User Access Keys with enough permissions.

Steps:

1. Open the PowerShell Window.
2. Change the directory to the above-created ec2-website-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. Run the `terraform output` command to get the values of defined variables in outputs.tf file.
9. Head to the **AWS Console**, and verify the created resources.

Route53 Configuration

Now we will configure the Route53 service for domain routing to the website.

Steps

1. Login to the AWS console and search for the **Route-53** service.
2. Click open the Route-53 console.
3. In the left plane of the window, click on `Hosted zones`.
4. Create a new hosted zone for your domain name e.g. `example.com`.
5. On completing, two records of type `NS` and `SOA` gets created here.
6. Copy all four values from column `Value/Route traffic to` of `NS` record.
7. Go to your domain provider's website and add these copied nameservers in place of your domain's original nameservers. This will dedicate your domain to AWS.
8. Again go to the hosted zone added in the AWS Route-53 console.
9. Add a `dev.example.com`(replace with your domain name) record of type `A` pointing to your EC2 instance's IP address received from running `terraform output` command.

SSH Into EC2 Server

Now we will SSH into the EC2 instance and configure the server for website deployment.

Steps

1. Open the Powershell Window.
2. Run the following command to SSH into EC2 server and substitute the <*private-ssh-key-path*> with the above generated private ssh key path and <*ec2-instance-public-ip*> with the server IP received from `terraform output` command:

```
    ssh -o StrictHostKeyChecking=no -i <*private-ssh-key-path*> ec2-user@<*ec2-
instance-public-ip*>
```

3. Once you enter the server, run the following commands to install the necessary dependencies for deployment:

```
    sudo yum update
    sudo yum install -y nodejs npm
    sudo npm install -g pm2
    sudo yum install -y nginx
    sudo yum install -y git
    sudo yum install -y certbot python3-certbot-nginx
    sudo npm install @sveltejs/adapter-node
```

4. Now clone the svelte git repository that you want to deploy by running and substituting the following link with your desired git repository link:

```
    git clone <https://github.com/account-name/repository-name.git>
```

5. Navigate to the website folder using the `cd` command and run the following commands to start the Nodejs server:

```
    sudo npm install @sveltejs/adapter-node
    sudo npm install
    sudo npm run build
    sudo pm2 start build/index.js --name "sveltekit-app"
    sudo pm2 save
    sudo pm2 startup
```

6. Now we will configure the Nginx as a Reverse Proxy with HTTPS to secure the web connection.
7. Click code and copy the default.conf file and paste it into the vim terminal opened by running the following command. Also, replace the <*dev.example.com*> with your desired domain name. Click `esc` and type `:wq` to write the file.
8. Run the following commands to configure the Nginx:

```
    sudo vim /etc/nginx/conf.d/default.conf
    sudo systemctl restart nginx
    sudo certbot --nginx -d dev.example.com
    sudo certbot renew --dry-run
```

9. Nginx configuration is complete. Try accessing the website on the browser.

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 **ec2-website-terraform** directory using the `cd` command.
2. Run `terraform destroy` & if prompted, type `yes`.
3. Infrastructure will be destroyed.

Deploy Static Website on CloudFront using Terraform

- We will provision the CloudFront for Static Website Delivery to users more quickly and securely.
- We will connect the CloudFront to ACM for secure web access.
- We will create S3 bucket to store the *static-website* files.
- We will also create a Route53 Hosted Zone and add a Route53 Record to access the website using the domain name.
- We will create all these resources using Terraform as an Infrastructure as Code.

Prerequisites

1. AWS Account with IAM User Access Keys
2. Terraform installed
3. Website repository
4. Domain name

Write Terraform Configuration files

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

Steps

1. Create the **cloudfront-website-terraform** project directory.
2. The folder structure for the above-created directory is as follows:

```
cloudfront-website-terraform
|───.terraform.lock.hcl
|───locals.tf
|───main.tf
|───providers.tf
|───terraform.tfstate
|───terraform.tfstate.backup
└───.terraform
```

> We need to only create *providers.tf*, *main.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

            - aws
5. Click code for reference.
6. The definition of *providers.tf* file is complete.
7. Now create the *main.tf* file.
8. Define the following modules inside it:
    - module.s3
    - module.acm-route53
    - module. cloudfront
    - module.route53-record
9. Click code for reference.
10. The definition of *main.tf* file is complete.
11. Now we will create *locals.tf* file.
12. Inside it, define the following variables:
    - local.aws-region
    - local.s3-properties
    - local.acm-properties
    - local.route53-zone-properties
    - local.cloudfront-properties
    - local.route53-record-properties
13. Click code for reference.
14. The definition of *locals.tf* file is complete.

> Ensure you give the appropriate values to the variables defined in *locals.tf* file.
> Also, ensure you set *aws-region* to **us-east-1**, as ACM will work with Cloudfront only & only if it is deployed in that region.

Provisioning the Infrastructure

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

> Ensure AWS CLI is configured with appropriate IAM User Access Keys with enough permissions.

Steps:

1. Open the PowerShell Window.
2. Change the directory to the above-created cloudfront-website-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. Apply will take time, till then
    1. Login to the AWS console and search for the **Route-53** service.
    2. Click open the Route-53 console.
    3. In the left plane of the window, click on `Hosted zones`.

    4. Terraform will create a new hosted zone for your domain e.g. `example.com`.
    5. On completing, two records of type `NS` and `SOA` gets created here.
    6. Copy all four values from column `Value/Route traffic to` of `NS` record.
    7. Go to your domain provider's website and add these copied nameservers in place of your domain's original nameservers. This will dedicate your domain to AWS.
    8. Now, head to the AWS Console and search for the **S3** service.
    9. Click open the **S3** service, select the terraform provisioned bucket, and upload the static website files to it.
    10. This will store all the website files and deliver them once the CloudFront requests.
9. Once the terraform provisioning is completed, try accessing the static website on the browser.

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 **cloudfront-website-terraform** directory using the `cd` command.
2. Run `terraform destroy` & if prompted, type `yes`.
3. Infrastructure will be destroyed.

ECS Provisioning using Terraform

- We will provision the ECS using Terraform as an Infrastructure as Code.
- We will deploy it in a custom Virtual Private Cloud for isolation.
- We will connect the Container App to ECR for Docker Image.
- We will also create S3 bucket to store the *.env* file.
- Also will deploy RDS MySQL Instance to store the relational data and connect it to ECS.

Prerequisites

---

1. AWS Account with an IAM User with administrative permissions.
2. Terraform installed.

Write Terraform Configuration files

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

Steps

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

```
ecs-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
        - docker
        - aws

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.vpc
    - module.s3
    - module.rds
    - module.ecr
    - module.load-balancer
    - module.ecs
9. Also define the following s3 resource for uploading local .env file:
    - resource.aws_s3_object
10. Click code for reference.
11. The definition of *main.tf* file is complete.
12. Now we will create *outputs.tf* file.
13. Inside it, define the following outputs.
    - output.DB_HOST
    - output.bastion-host-ip
14. Click code for reference.
15. The definition of *outputs.tf* file is complete.
16. Now we will create *locals.tf* file.
17. Inside it, define the following variables:
    - local.vpc-properties
    - local.s3-properties
    - local.database-properties
    - local.bastion-properties
    - local.load-balancer-properties
    - local.ecs-properties
18. Click code for reference.
19. 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 *s3-object-source-path* variable under *s3-properties* with local *.env* file relative path.

Provisioning the Infrastructure

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

> Ensure AWS CLI is configured with appropriate AWS user credentials and enough permissions.

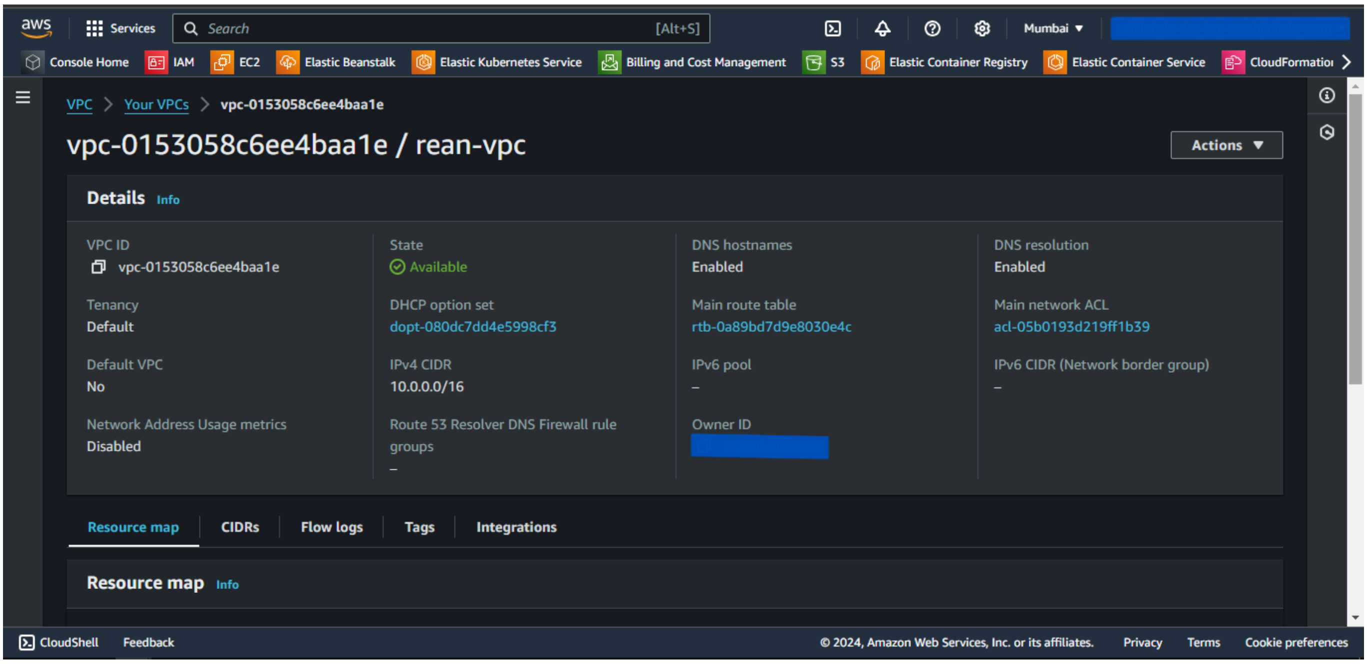
Steps:

1. Open the PowerShell.
2. Change the directory to the above-created **ecs-terraform** directory using `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. Run the `terraform output` command to get the values of defined variables in *outputs.tf* file.
9. Head to the AWS Console, and verify the created resources.
10. Then,
    - Head towards EC2 dashboard.
    - Select *Load Balancers*, and select the created load balancer.
    - Copy the DNS address.
    - Paste the address in the browser to access the application.

Screenshots of Provisioned Infrastructure

VPC Image


aws
Services
Search
[Alt+S]
Mumbai ▼
[redacted]
Console Home
IAM
EC2
Elastic Beanstalk
Elastic Kubernetes Service
Billing and Cost Management
S3
Elastic Container Registry
Elastic Container Service
CloudFormatio
VPC > Your VPCs > vpc-0153058c6ee4baa1e
Actions ▼
Resource map
CIDRs
Flow logs
Tags
Integrations
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

vpc-0153058c6ee4baa1e / rean-vpc

Details Info

VPC ID: vpc-0153058c6ee4baa1e

State: Available

DNS hostnames: Enabled

DNS resolution: Enabled

Tenancy: Default

DHCP option set: dopt-080dc7dd4e5998cf3

Main route table: rtb-0a89bd7d9e8030e4c

Main network ACL: acl-05b0193d219ff1b39

Default VPC: No

IPv4 CIDR: 10.0.0.0/16

IPv6 pool: –

IPv6 CIDR (Network border group): –

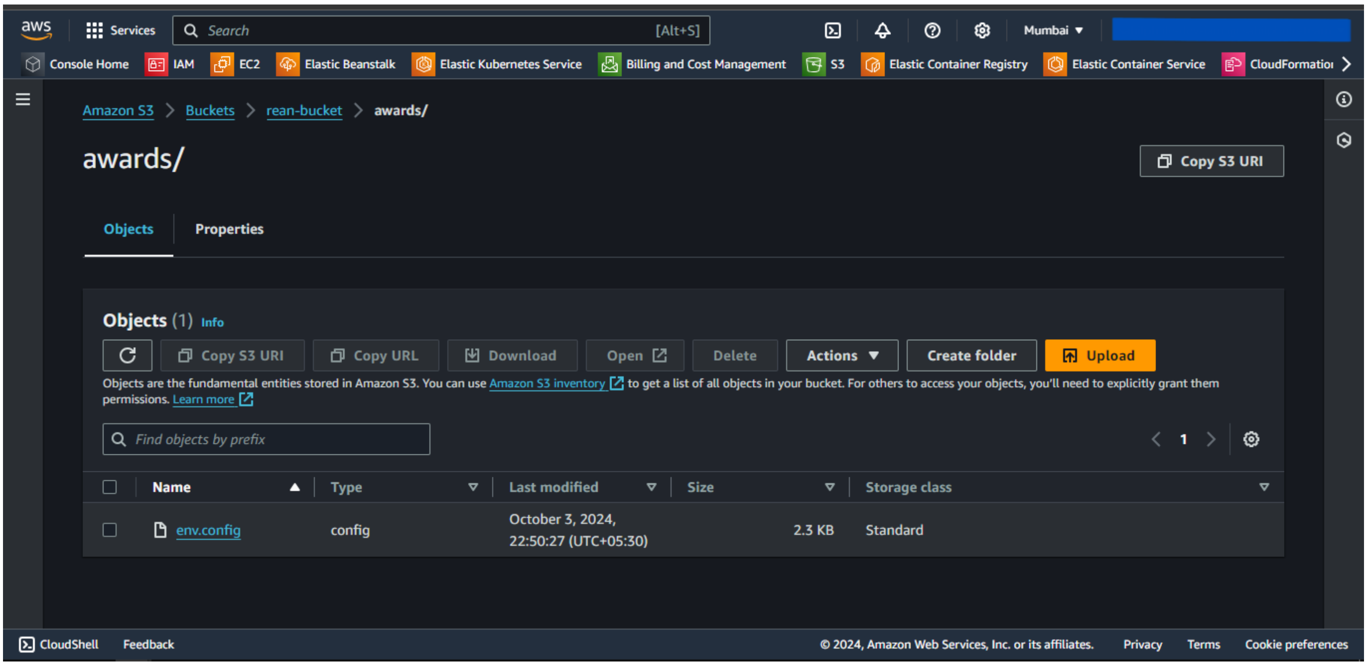
Network Address Usage metrics: Disabled

Route 53 Resolver DNS Firewall rule groups: –

Owner ID: [redacted]

Resource map Info

S3 Image



Amazon S3 > Buckets > rean-bucket > awards/

awards/

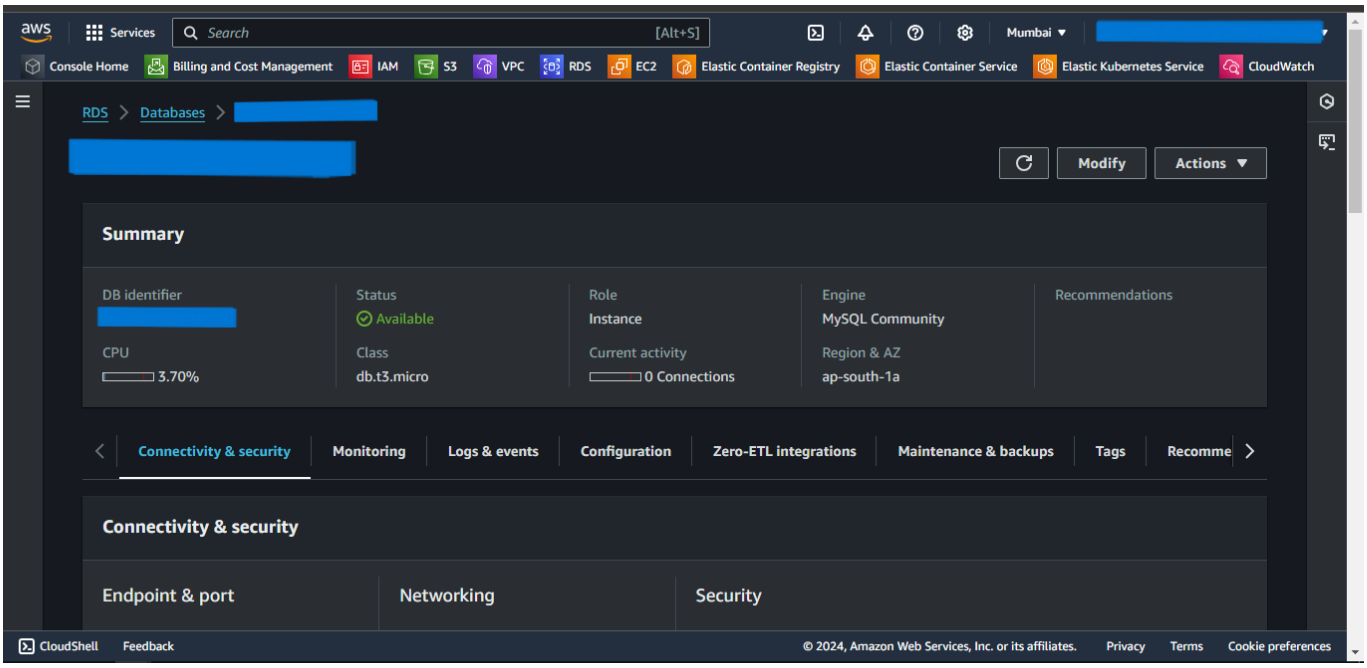
Objects (1) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	env.config	config	October 3, 2024, 22:50:27 (UTC+05:30)	2.3 KB	Standard

---

RDS Image



RDS > Databases > [redacted]

[redacted]

Summary

DB identifier	Status	Role	Engine	Recommendations
[redacted]	✓ Available	Instance	MySQL Community	
CPU	Class	Current activity	Region & AZ	
3.70%	db.t3.micro	0 Connections	ap-south-1a	

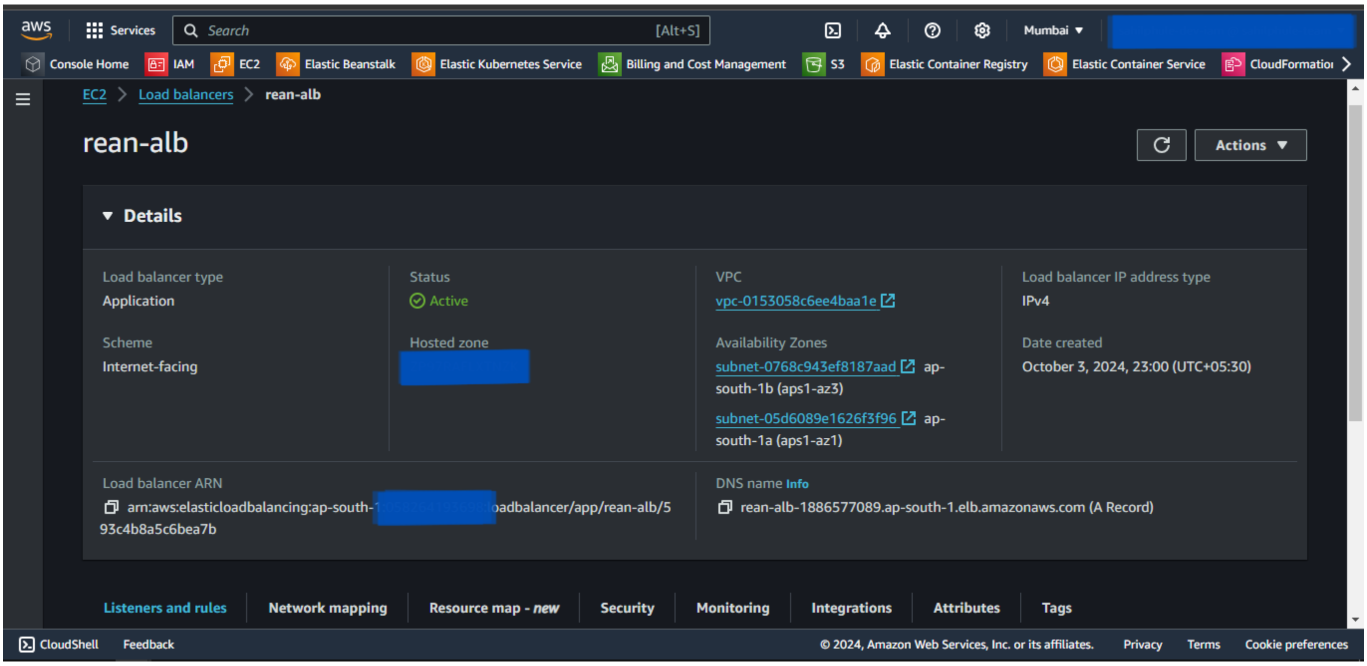
Connectivity & security

Endpoint & port

Networking

Security

ALB Image



rean-alb

Load balancer type
Application

Status
Active

VPC
vpc-0153058c6ee4baa1e

Load balancer IP address type
IPv4

Scheme
Internet-facing

Hosted zone
[redacted]

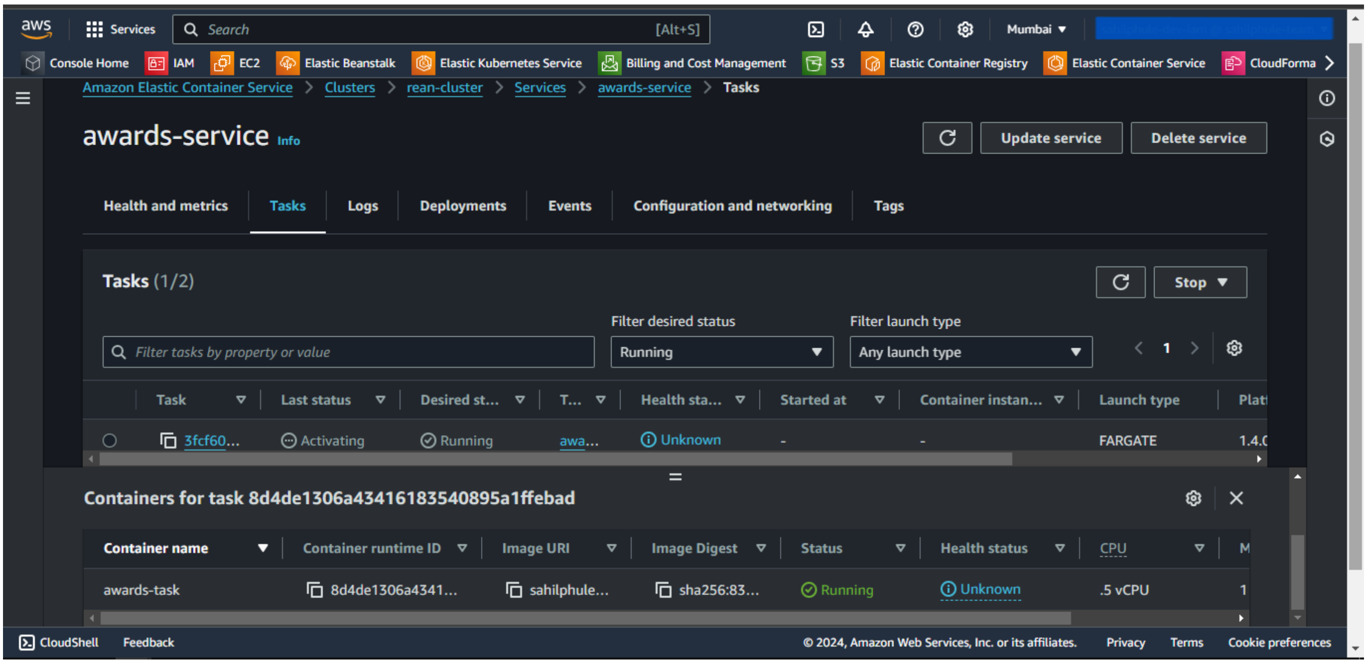
Availability Zones
subnet-0768c943ef8187aad ap-south-1b (aps1-az3)
subnet-05d6089e1626f3f96 ap-south-1a (aps1-az1)

Date created
October 3, 2024, 23:00 (UTC+05:30)

Load balancer ARN
arn:aws:elasticloadbalancing:ap-south-[redacted]oadbalancer/app/rean-alb/593c4b8a5c6bea7b

DNS name Info
rean-alb-1886577089.ap-south-1.elb.amazonaws.com (A Record)

ECS Image


aws
Services
Search [Alt+S]
Mumbai ▾
[redacted]
Console Home
IAM
EC2
Elastic Beanstalk
Elastic Kubernetes Service
Billing and Cost Management
S3
Elastic Container Registry
Elastic Container Service
CloudForma
Amazon Elastic Container Service > Clusters > rean-cluster > Services > awards-service > Tasks
Update service
Delete service
Health and metrics
Tasks
Logs
Deployments
Events
Configuration and networking
Tags
Stop ▼
Filter tasks by property or value
< 1 >
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

awards-service Info

Tasks (1/2)

Filter desired status: Running

Filter launch type: Any launch type

	Task	Last status	Desired st...	T...	Health sta...	Started at	Container instan...	Launch type	Plat
○	3fcf60...	Activating	Running	awa...	Unknown	-	-	FARGATE	1.4.(

Containers for task 8d4de1306a43416183540895a1ffebad

Container name	Container runtime ID	Image URI	Image Digest	Status	Health status	CPU	M
awards-task	8d4de1306a4341...	sahilphule...	sha256:83...	Running	Unknown	.5 vCPU	1

Connection to the RDS database through Bastion Host using MySQL Workbench

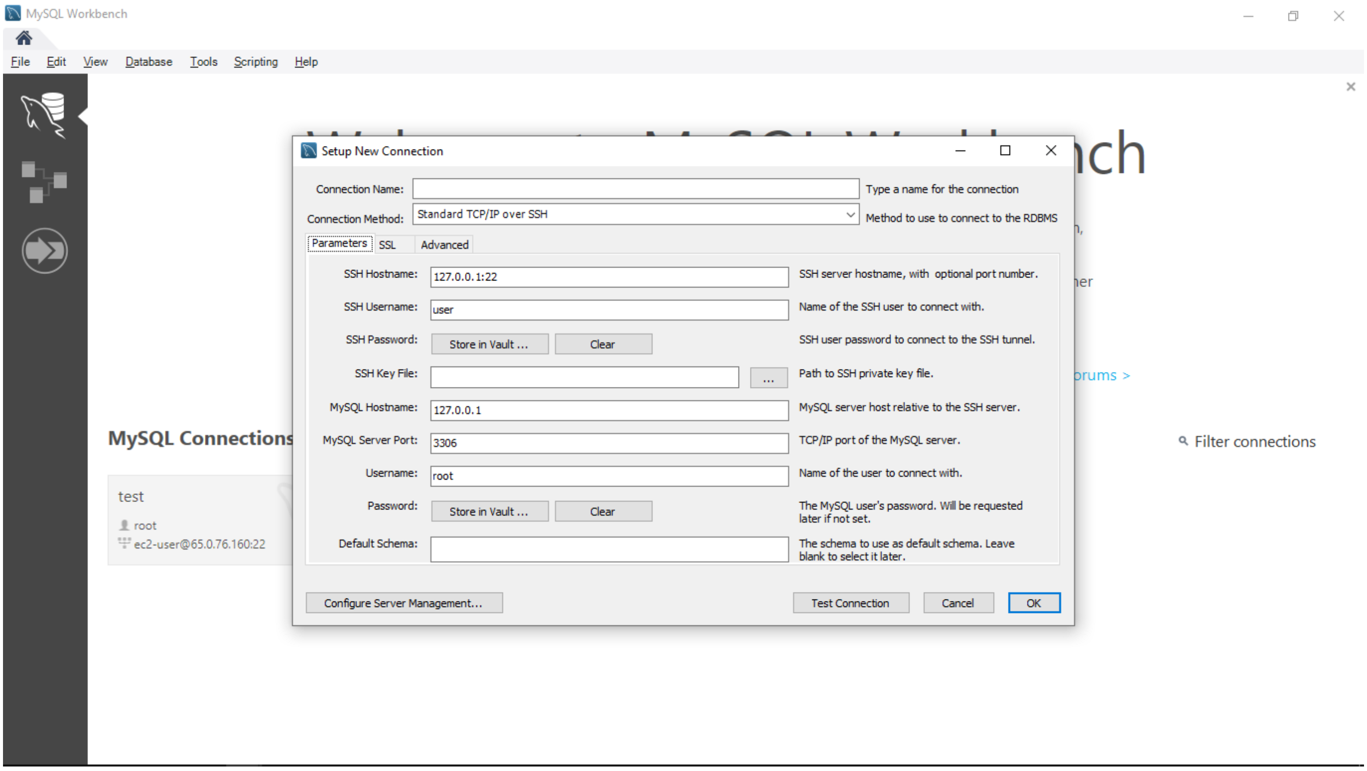
Now, we will use MySQL Workbench to connect and access the MySQL RDS Database through above created Bastion Host.

Steps

1. Open MySQL Workbench.
2. Click Add Connection.
3. Select connection method as **Standard TCP/IP over SSH**.
4. In SSH Hostname, enter *bastion-host-ip:22* where bastion-host-ip is received from the `terraform output` command.
5. In SSH Username, enter *ec2-user*.
6. In SSH Key File, select *bastion-key.pem* file passed in above *locals.tf* file from your local computer.
7. In MySQL Hostname, enter *DB_HOST* where DB_HOST is received from the `terraform output` command.
8. In the Password section, select *Store in Vault*, and enter the password passed in above-created *locals.tf* file.
9. Click *OK* and open the connection.
10. Now you can run MySQL commands to access databases and verify the successful connection of *ecs-container*.

Screenshots of MySQL Workbench

Connection Page



Setup New Connection

Connection Name: — Type a name for the connection

Connection Method: Standard TCP/IP over SSH — Method to use to connect to the RDBMS

SSH Hostname: 127.0.0.1:22 — SSH server hostname, with optional port number.

SSH Username: user — Name of the SSH user to connect with.

SSH Password: — SSH user password to connect to the SSH tunnel.

SSH Key File: — Path to SSH private key file.

MySQL Hostname: 127.0.0.1 — MySQL server host relative to the SSH server.

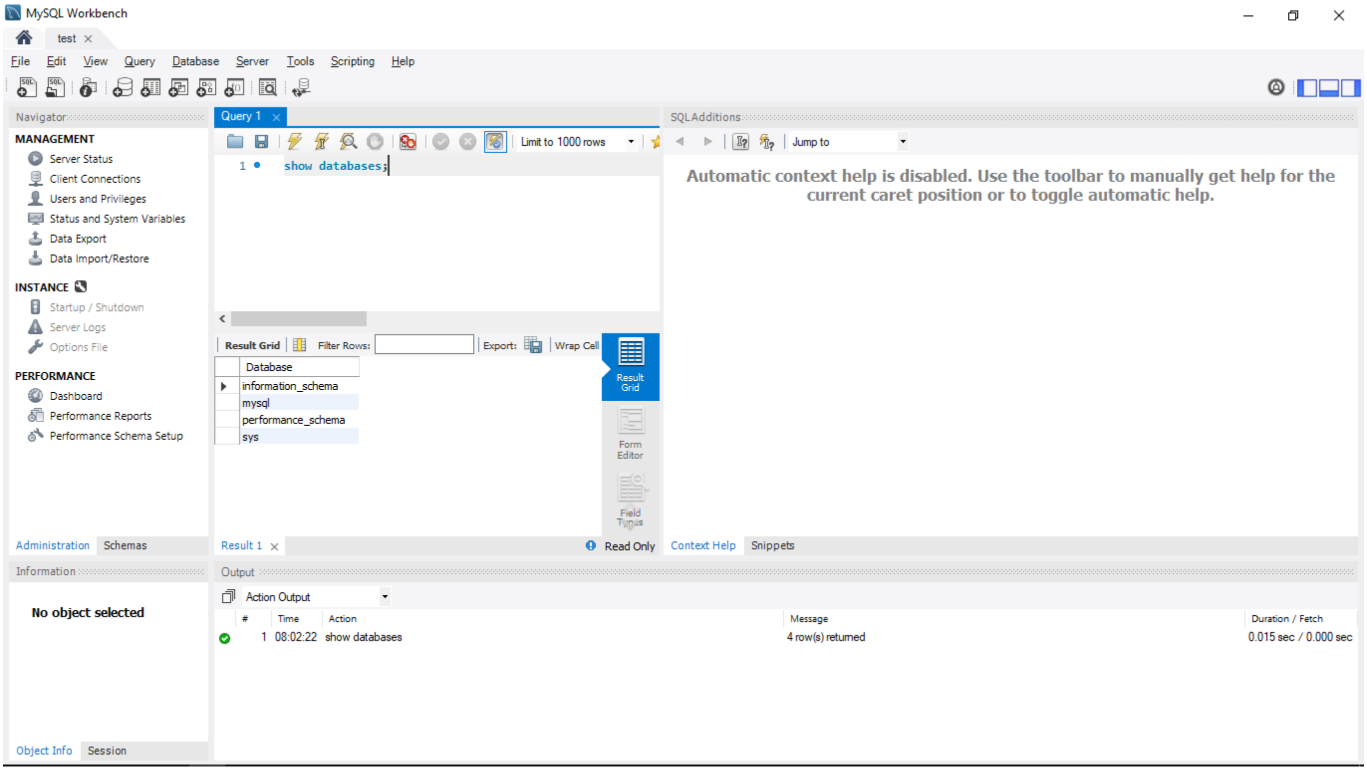
MySQL Server Port: 3306 — TCP/IP port of the MySQL server.

Username: root — Name of the user to connect with.

Password: — The MySQL user's password. Will be requested later if not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

Commands Page



```
show databases;
```

Database
information_schema
mysql
performance_schema
sys

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

No object selected

#	Time	Action	Message	Duration / Fetch
1	08:02:22	show databases	4 row(s) returned	0.015 sec / 0.000 sec

---

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 **ecs-terraform** directory using the `cd` command.
2. Run `terraform destroy` & if prompted, type `yes`.
3. Infrastructure will be destroyed.

EKS Provisioning using Terraform

- We will provision the EKS using Terraform as an Infrastructure as Code.
- We will deploy it in a custom Virtual Private Cloud for isolation.
- We will also deploy RDS MySQL Instance to store the relational data and connect it to EKS.

Prerequisites

---

1. AWS Account with an IAM User with administrative permissions.
2. Terraform installed.
3. Kubectl installed.

Write Terraform Configuration files

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

Steps

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

```
eks-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
        - aws
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:
    - vpc
    - rds
    - eks
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.
    - DB_HOST
    - bastion-host-ip
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:
    - vpc-properties
    - database-properties
    - bastion-properties
    - eks-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.

Provisioning the Infrastructure

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

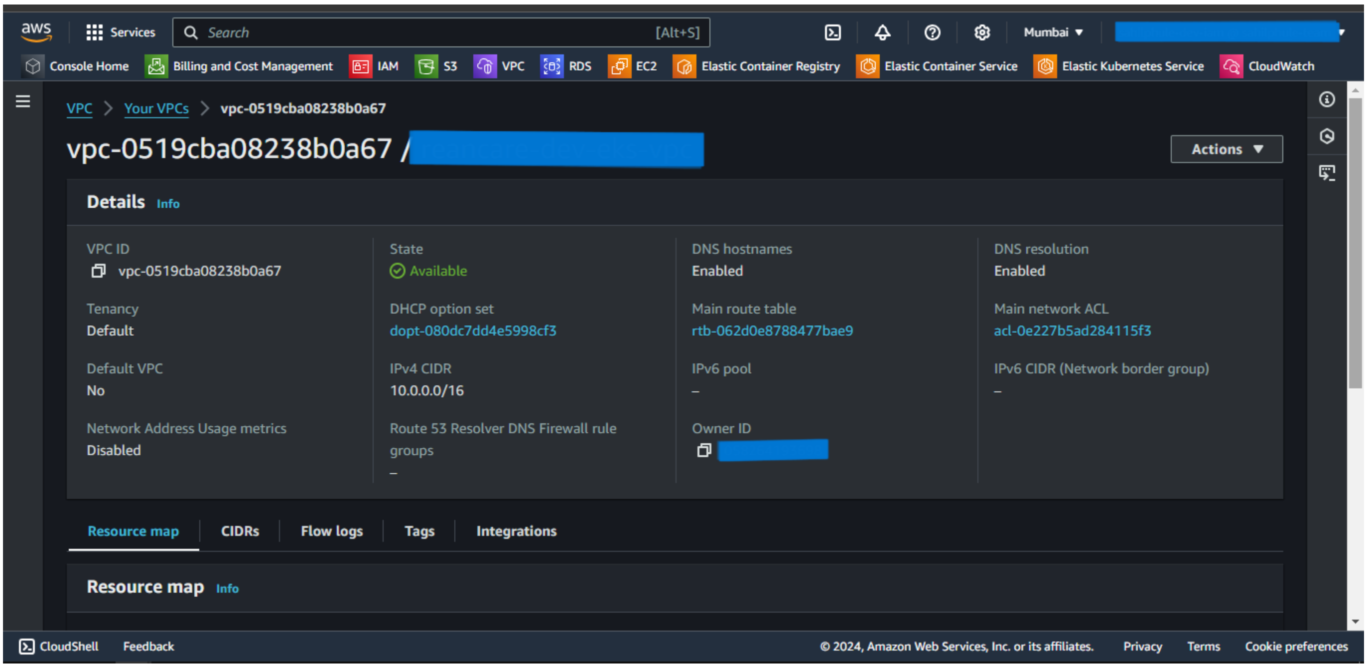
> Ensure AWS CLI is configured with appropriate AWS user credentials and enough permissions.

Steps:

1. Open the PowerShell.
2. Change the directory to the above-created **eks-terraform** directory using `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. Run the `terraform output` command to get the values of defined variables in *outputs.tf* file.
9. Head to the AWS Console, and verify the created resources.

Screenshots of Provisioned Infrastructure

VPC Image



VPC > Your VPCs > vpc-0519cba08238b0a67

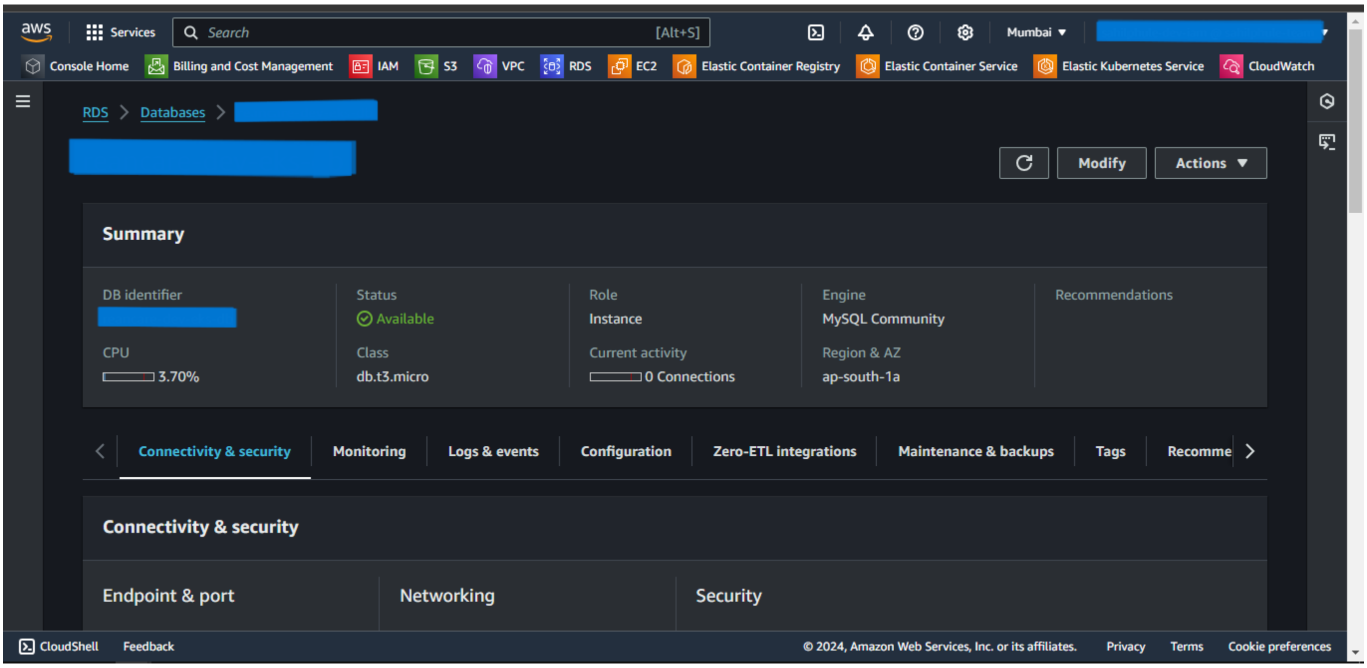
vpc-0519cba08238b0a67 / [redacted]

Details Info

VPC ID: vpc-0519cba08238b0a67
State: Available
DNS hostnames: Enabled
DNS resolution: Enabled
Tenancy: Default
DHCP option set: dopt-080dc7dd4e5998cf3
Main route table: rtb-062d0e8788477bae9
Main network ACL: acl-0e227b5ad284115f3
Default VPC: No
IPv4 CIDR: 10.0.0.0/16
IPv6 pool: –
IPv6 CIDR (Network border group): –
Network Address Usage metrics: Disabled
Route 53 Resolver DNS Firewall rule groups: –
Owner ID: [redacted]

Resource map Info

RDS Image



RDS > Databases > [redacted]

[redacted]

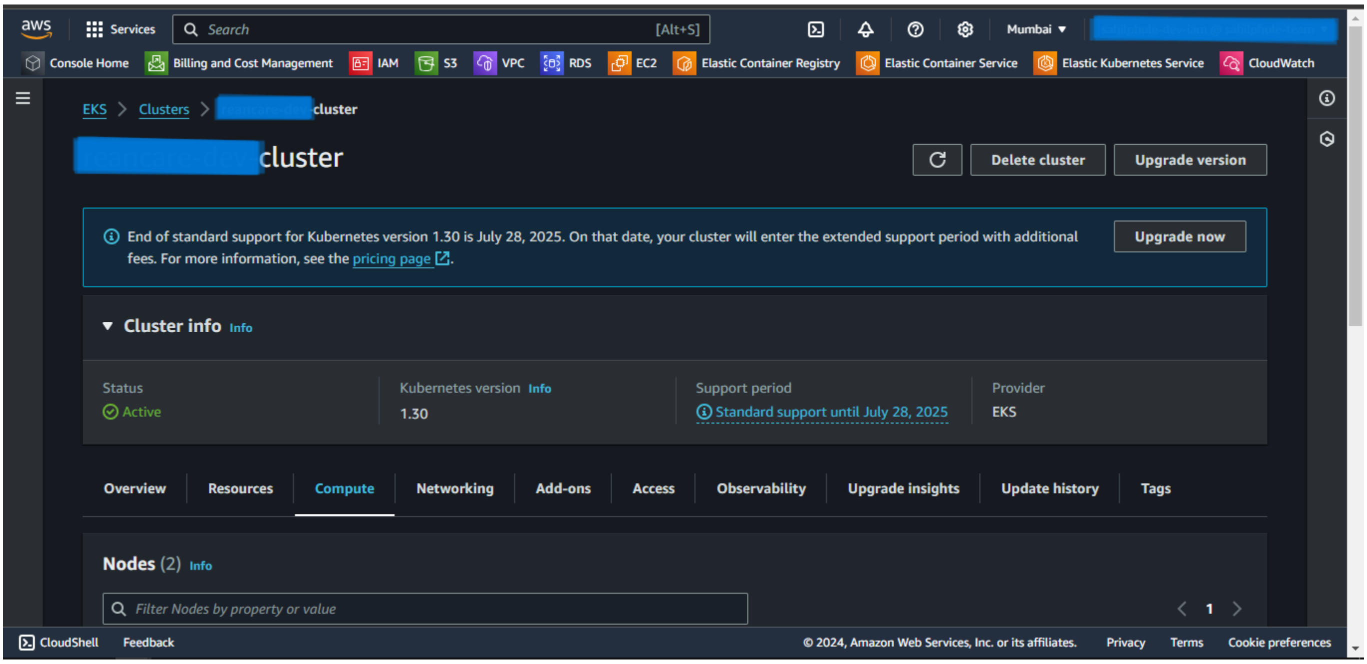
Summary

DB identifier: [redacted]
Status: Available
Role: Instance
Engine: MySQL Community
Recommendations
CPU: 3.70%
Class: db.t3.micro
Current activity: 0 Connections
Region & AZ: ap-south-1a

Connectivity & security

Endpoint & port | Networking | Security

EKS Cluster Image



[redacted]

EKS > Clusters > [redacted]cluster

[redacted]cluster

End of standard support for Kubernetes version 1.30 is July 28, 2025. On that date, your cluster will enter the extended support period with additional fees. For more information, see the pricing page.

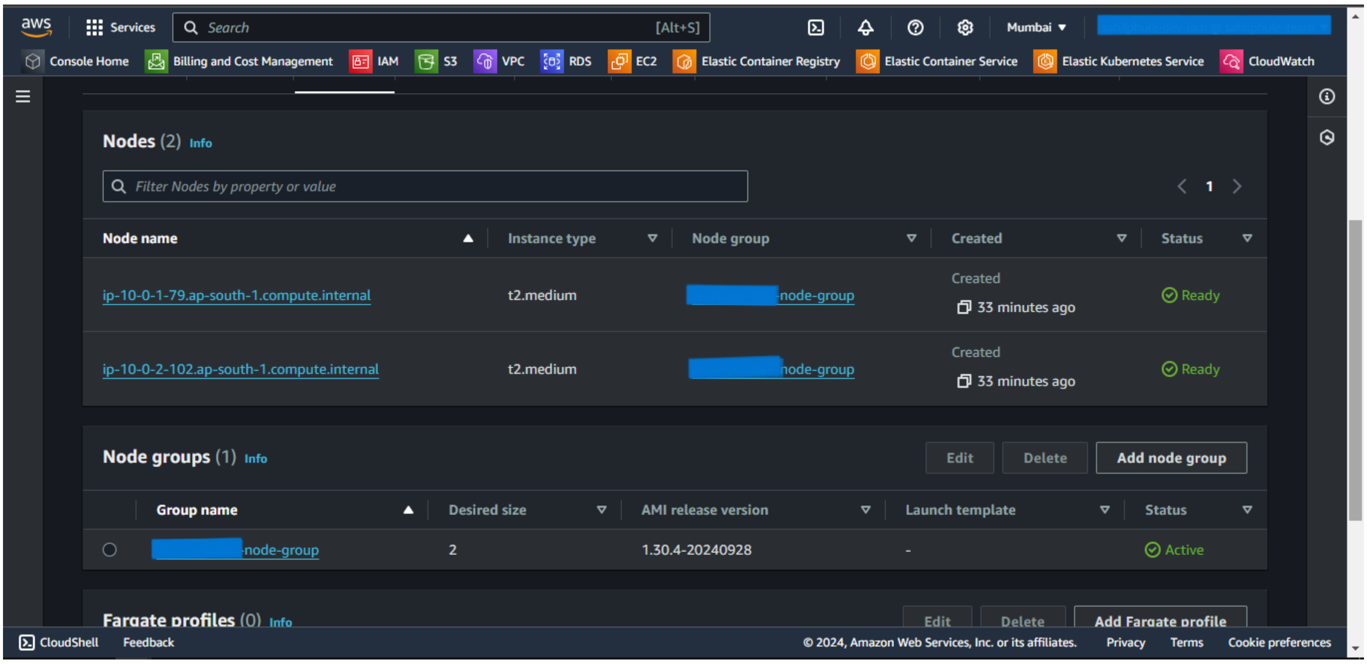
Cluster info Info

Status	Kubernetes version Info	Support period	Provider
Active	1.30	Standard support until July 28, 2025	EKS

Nodes (2) Info

Filter Nodes by property or value

EKS Node Group Image



[redacted]

Nodes (2) Info

Filter Nodes by property or value

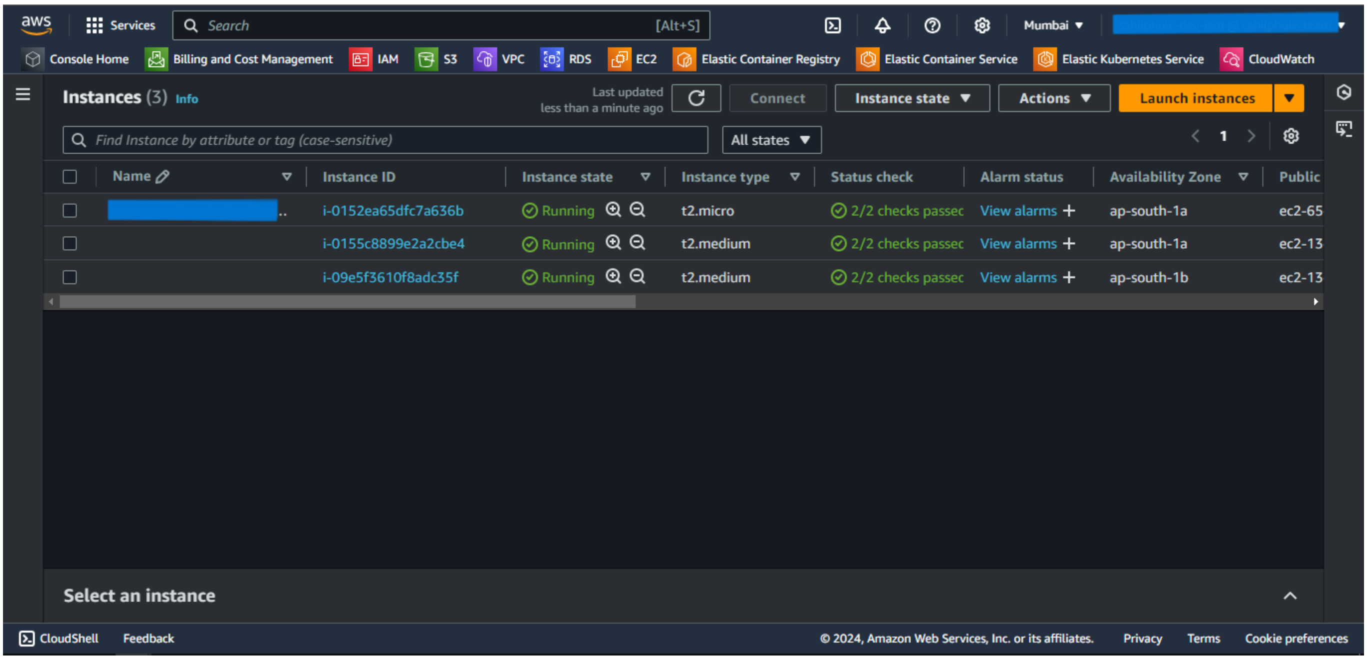
Node name	Instance type	Node group	Created	Status
ip-10-0-1-79.ap-south-1.compute.internal	t2.medium	[redacted]node-group	Created 33 minutes ago	Ready
ip-10-0-2-102.ap-south-1.compute.internal	t2.medium	[redacted]node-group	Created 33 minutes ago	Ready

Node groups (1) Info

	Group name	Desired size	AMI release version	Launch template	Status
○	[redacted]node-group	2	1.30.4-20240928	-	Active

Fargate profiles (0) Info

EKS Nodes Image



Instances (3) Info

Last updated less than a minute ago

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public
□	[redacted]..	i-0152ea65dfc7a636b	Running	t2.micro	2/2 checks passec	View alarms +	ap-south-1a	ec2-65
□		i-0155c8899e2a2cbe4	Running	t2.medium	2/2 checks passec	View alarms +	ap-south-1a	ec2-13
□		i-09e5f3610f8adc35f	Running	t2.medium	2/2 checks passec	View alarms +	ap-south-1b	ec2-13

Connect to the EKS Cluster from Powershell

Steps

1. Open a new Powershell window.
2. Run the following command to configure local kubectl with eks cluster:

```
aws eks --region <region-name> update-kubeconfig --name <cluster-name>
```

> Substitute <*region-name*> and <*cluster-name*> with the values defined in the above-created *locals.tf* file.

3. Now apply the Kubernetes manifest files of the application using the following command:

```
kubectl apply -f <file-path>
```

> Substitute <*file-path*> with the Kubernetes manifest file path.

4. To list them all, run `kubectl get all`.
5. If a Load Balancer type Service is present then try accessing the External IP of that service in the browser.

Connection to the RDS database through Bastion Host using MySQL Workbench

---

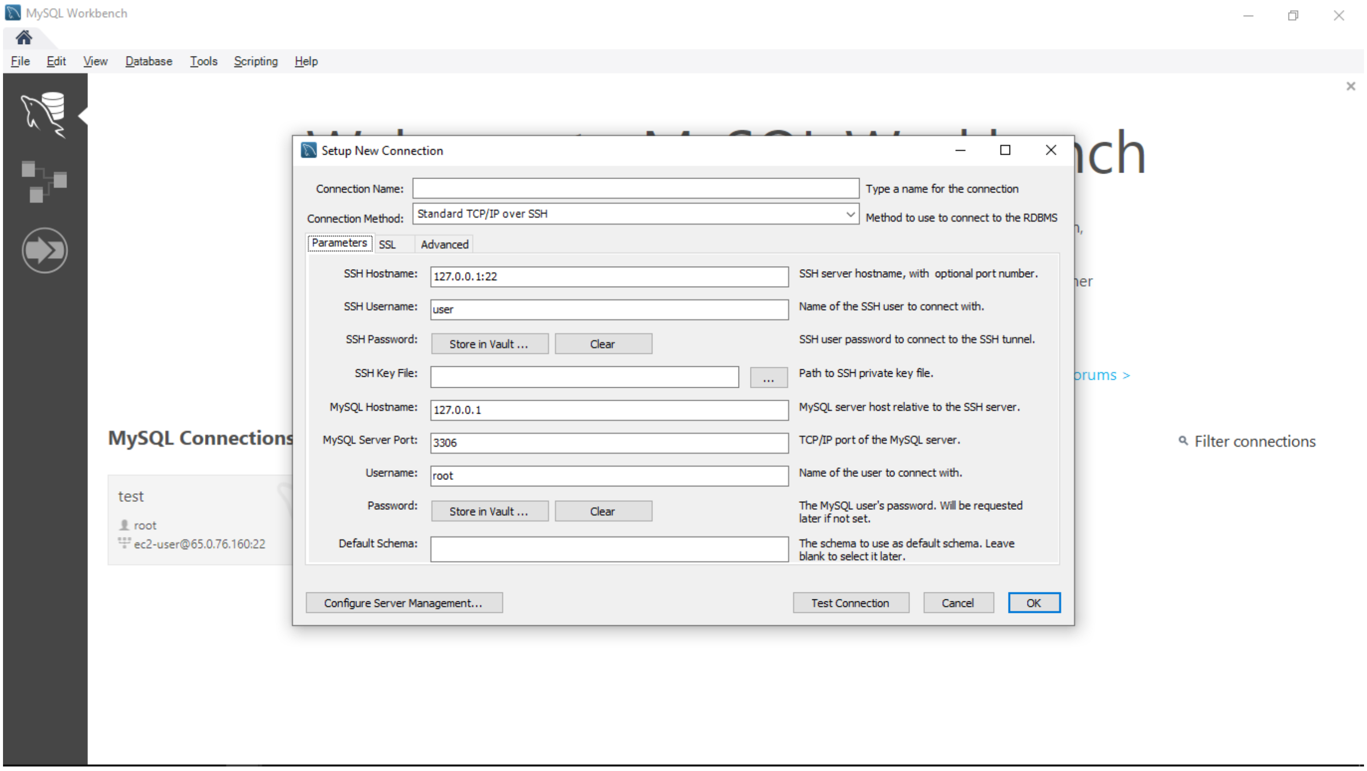
Now, we will use MySQL Workbench to connect and access the MySQL RDS Database through above created Bastion Host.

Steps

1. Open MySQL Workbench.
2. Click Add Connection.
3. Select connection method as **Standard TCP/IP over SSH**.
4. In SSH Hostname, enter *bastion-host-ip:22* where bastion-host-ip is received from the `terraform output` command.
5. In SSH Username, enter *ec2-user*.
6. In SSH Key File, select *bastion-key.pem* file passed in above *locals.tf* file from your local computer.
7. In MySQL Hostname, enter *DB_HOST* where DB_HOST is received from the `terraform output` command.
8. In the Password section, select *Store in Vault*, and enter the password passed in above-created *locals.tf* file.
9. Click *OK* and open the connection.
10. Now you can run MySQL commands to access databases and verify the successful connection of *eks-nodes*.

Screenshots of MySQL Workbench

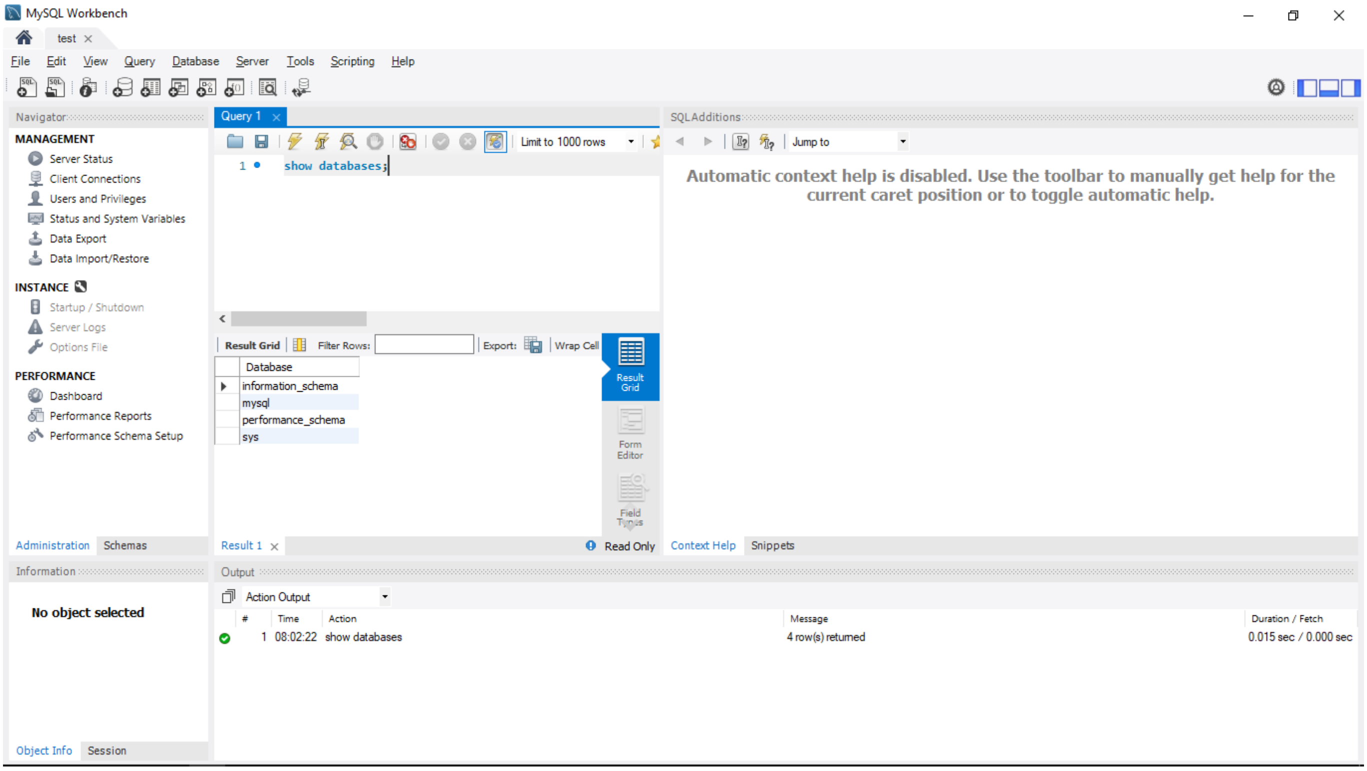
Connection Page



Setup New Connection

Connection Name:		Type a name for the connection
Connection Method:	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname:	127.0.0.1:22	SSH server hostname, with optional port number.
SSH Username:	user	Name of the SSH user to connect with.
SSH Password:	Store in Vault ... Clear	SSH user password to connect to the SSH tunnel.
SSH Key File:		Path to SSH private key file.
MySQL Hostname:	127.0.0.1	MySQL server host relative to the SSH server.
MySQL Server Port:	3306	TCP/IP port of the MySQL server.
Username:	root	Name of the user to connect with.
Password:	Store in Vault ... Clear	The MySQL user's password. Will be requested later if not set.
Default Schema:		The schema to use as default schema. Leave blank to select it later.

Commands Page



```
show databases;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Database
information_schema
mysql
performance_schema
sys

#	Time	Action	Message	Duration / Fetch
1	08:02:22	show databases	4 row(s) returned	0.015 sec / 0.000 sec

---

Destroy the provisioned infrastructure

Lastly, we will destroy the above-created resources.

Steps

1. Firstly, delete all the Kubernetes Deployments using:

```
kubectl delete -f <file-path>
```

> Substitute *<file-path>* with the Kubernetes manifest file path.

2. To destroy infrastructure, change the directory to the above-created **eks-terraform** directory using the `cd` command.
3. Run `terraform destroy` & if prompted, type `yes`.
4. Infrastructure will be destroyed.

Azure Virtual Machine Provisioning using Terraform

- We will provision the Azure Virtual Machine using Terraform as an Infrastructure as Code.
- We will deploy it in a custom Virtual Network for isolation.
- We will SSH into the Virtual Machine, and install the docker.
- Then, we will deploy the Nginx Container and try accessing it on the Web Browser.

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 **virtual-machine-terraform** directory.
2. The folder structure for the above-created directory is as follows:

```
virtual-machine-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* files. 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.virtual-machine
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.vm-public-ip
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.virtual-machine-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.

Provisioning the Infrastructure

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

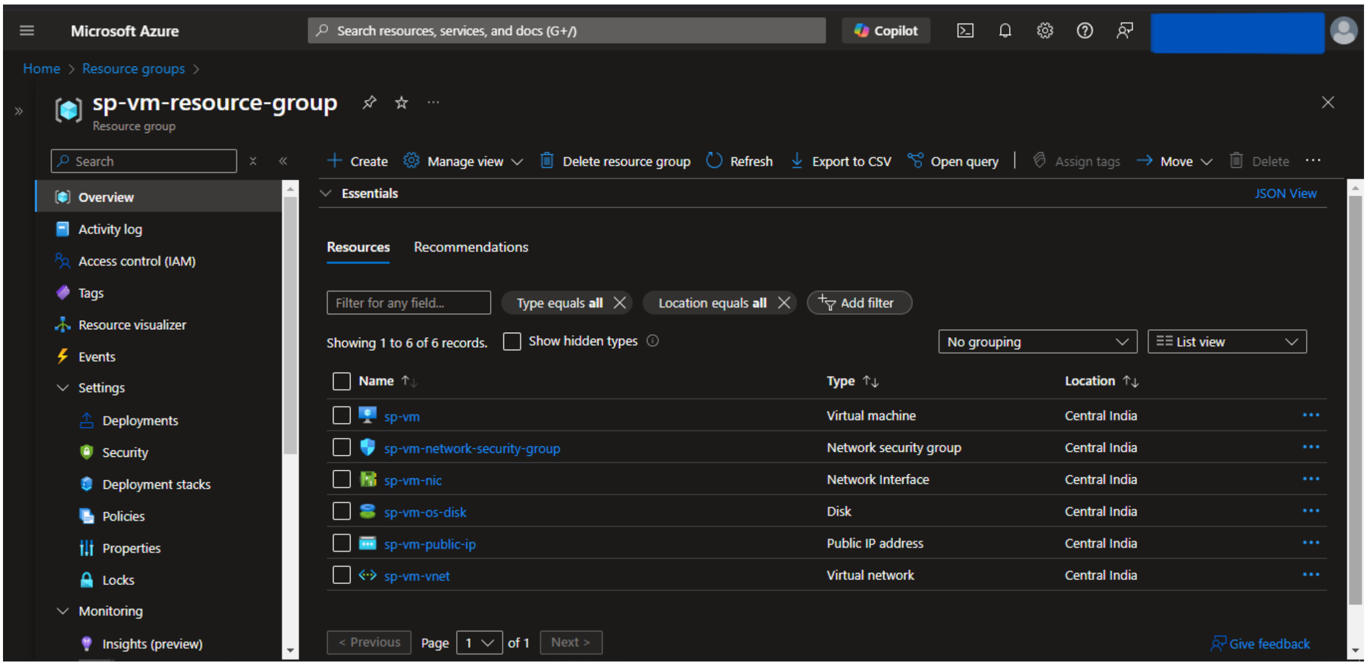
> Ensure Azure CLI is configured with appropriate Azure Account credentials and enough permissions.

Steps:

1. Open the PowerShell Window.
2. Change the directory to the above-created **virtual-machine-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. Run the `terraform output` command to get the values of defined variables in *outputs.tf* file.
9. Head to the Azure Console, and verify the created resources.

Screenshots of Provisioned Infrastructure

Resource Group Image



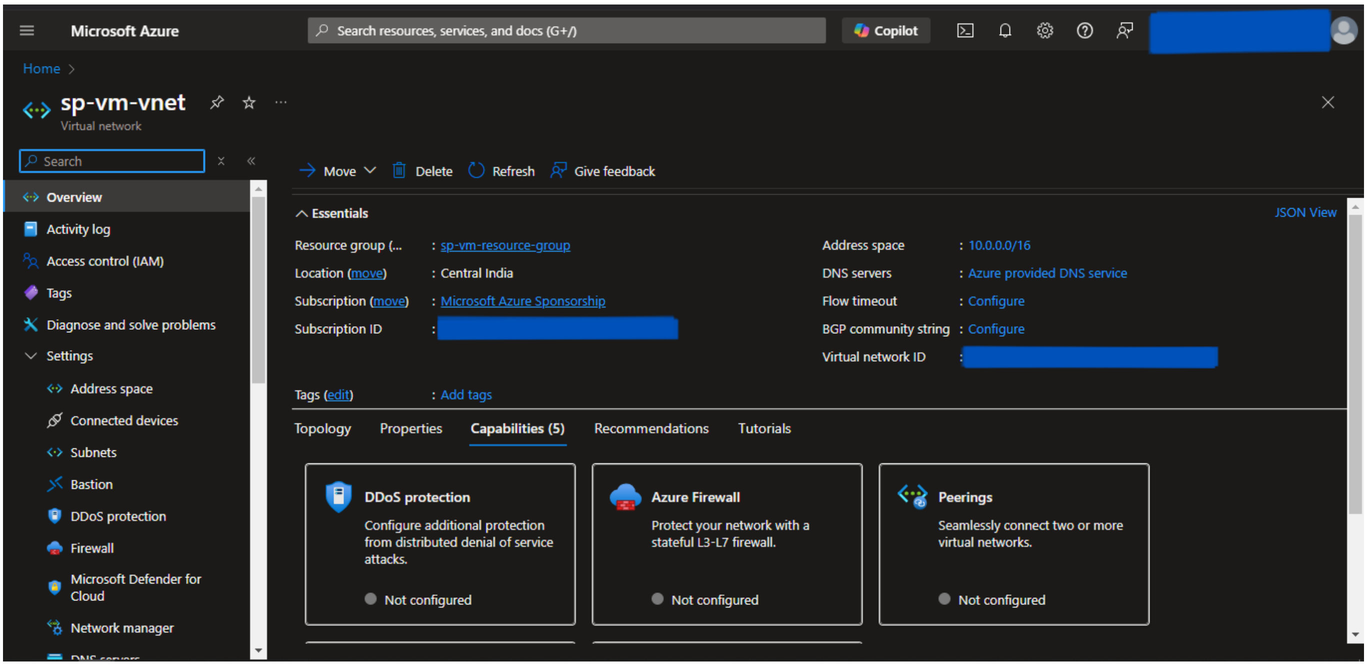
sp-vm-resource-group

Type equals **all** · Location equals **all**

Showing 1 to 6 of 6 records. □ Show hidden types

Name	Type	Location
sp-vm	Virtual machine	Central India
sp-vm-network-security-group	Network security group	Central India
sp-vm-nic	Network Interface	Central India
sp-vm-os-disk	Disk	Central India
sp-vm-public-ip	Public IP address	Central India
sp-vm-vnet	Virtual network	Central India

Virtual Network Image



sp-vm-vnet

Resource group (...	: sp-vm-resource-group	Address space	: 10.0.0.0/16
Location (move)	: Central India	DNS servers	: Azure provided DNS service
Subscription (move)	: Microsoft Azure Sponsorship	Flow timeout	: Configure
Subscription ID	: [redacted]	BGP community string	: Configure
		Virtual network ID	: [redacted]

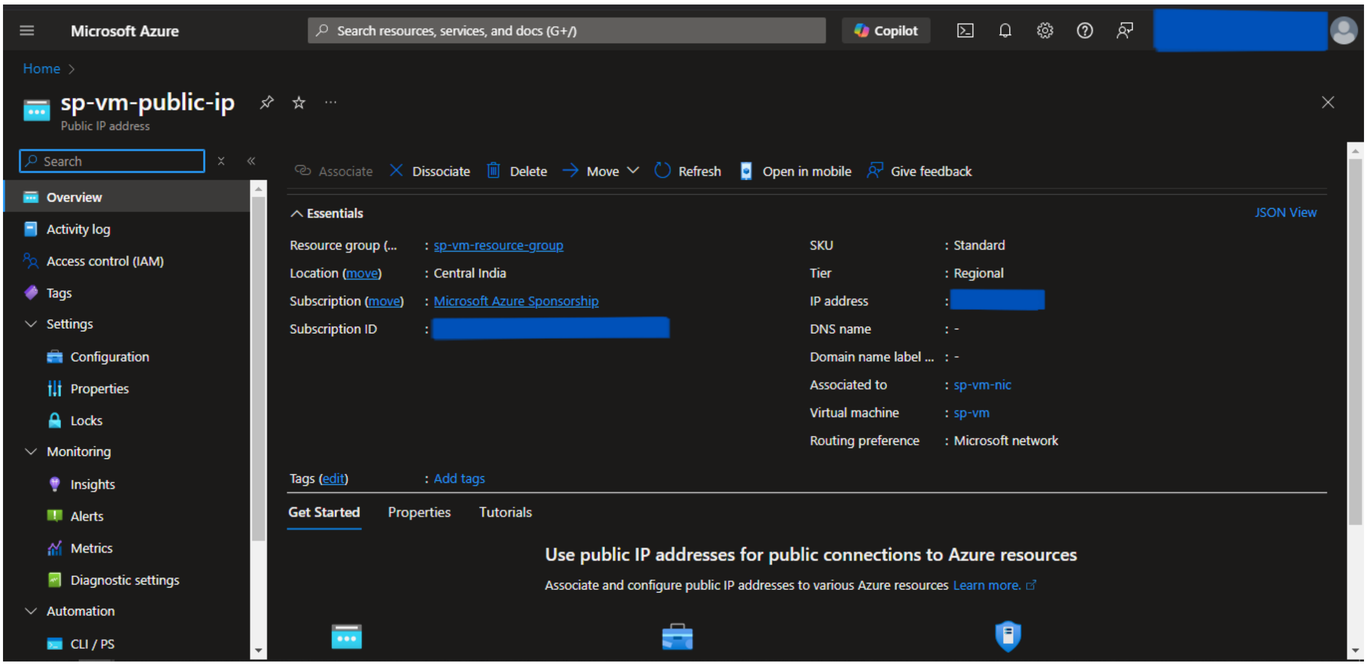
Tags (edit) : Add tags

DDoS protection
Configure additional protection from distributed denial of service attacks.
● Not configured

Azure Firewall
Protect your network with a stateful L3-L7 firewall.
● Not configured

Peerings
Seamlessly connect two or more virtual networks.
● Not configured

Public IP Image



sp-vm-public-ip

Public IP address

Essentials

Resource group (... : sp-vm-resource-group

Location (move) : Central India

Subscription (move) : Microsoft Azure Sponsorship

Subscription ID : [redacted]

SKU : Standard

Tier : Regional

IP address : [redacted]

DNS name : -

Domain name label ... : -

Associated to : sp-vm-nic

Virtual machine : sp-vm

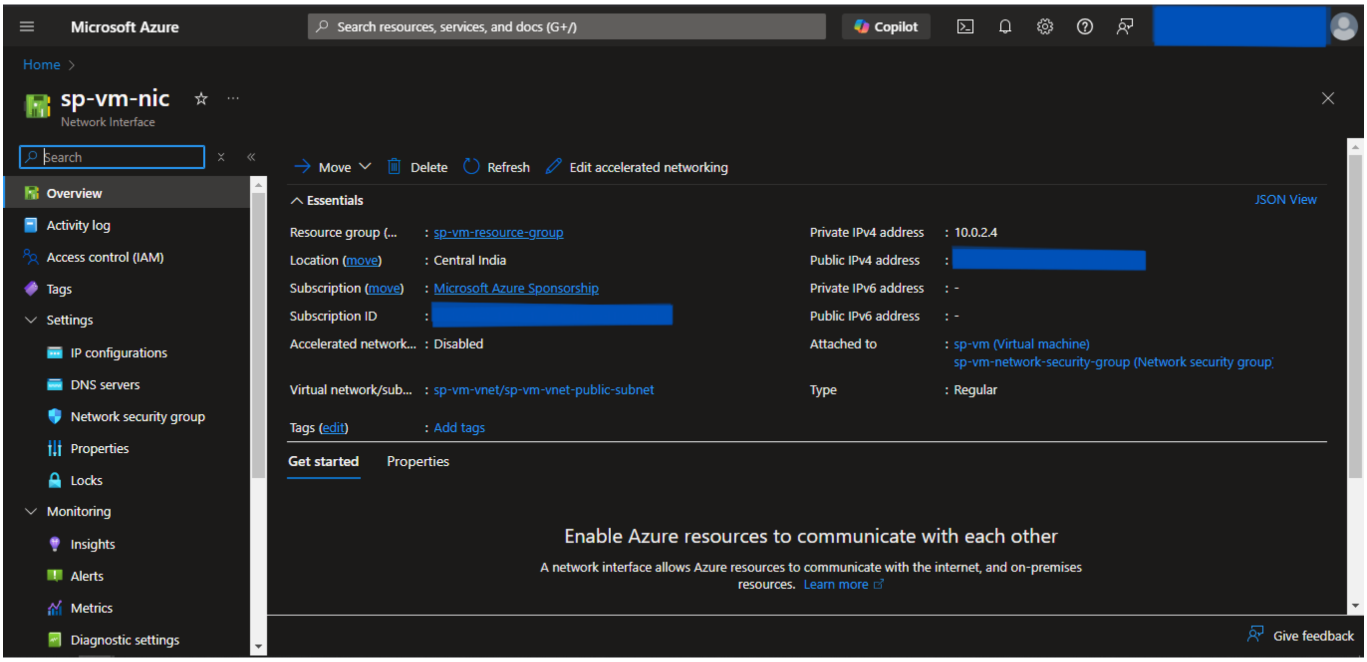
Routing preference : Microsoft network

Tags (edit) : Add tags

Use public IP addresses for public connections to Azure resources

Associate and configure public IP addresses to various Azure resources Learn more.

Network Interface Card Image



sp-vm-nic

Network Interface

Essentials

Resource group (... : sp-vm-resource-group

Location (move) : Central India

Subscription (move) : Microsoft Azure Sponsorship

Subscription ID : [redacted]

Accelerated network... : Disabled

Virtual network/sub... : sp-vm-vnet/sp-vm-vnet-public-subnet

Private IPv4 address : 10.0.2.4

Public IPv4 address : [redacted]

Private IPv6 address : -

Public IPv6 address : -

Attached to : sp-vm (Virtual machine)
sp-vm-network-security-group (Network security group)

Type : Regular

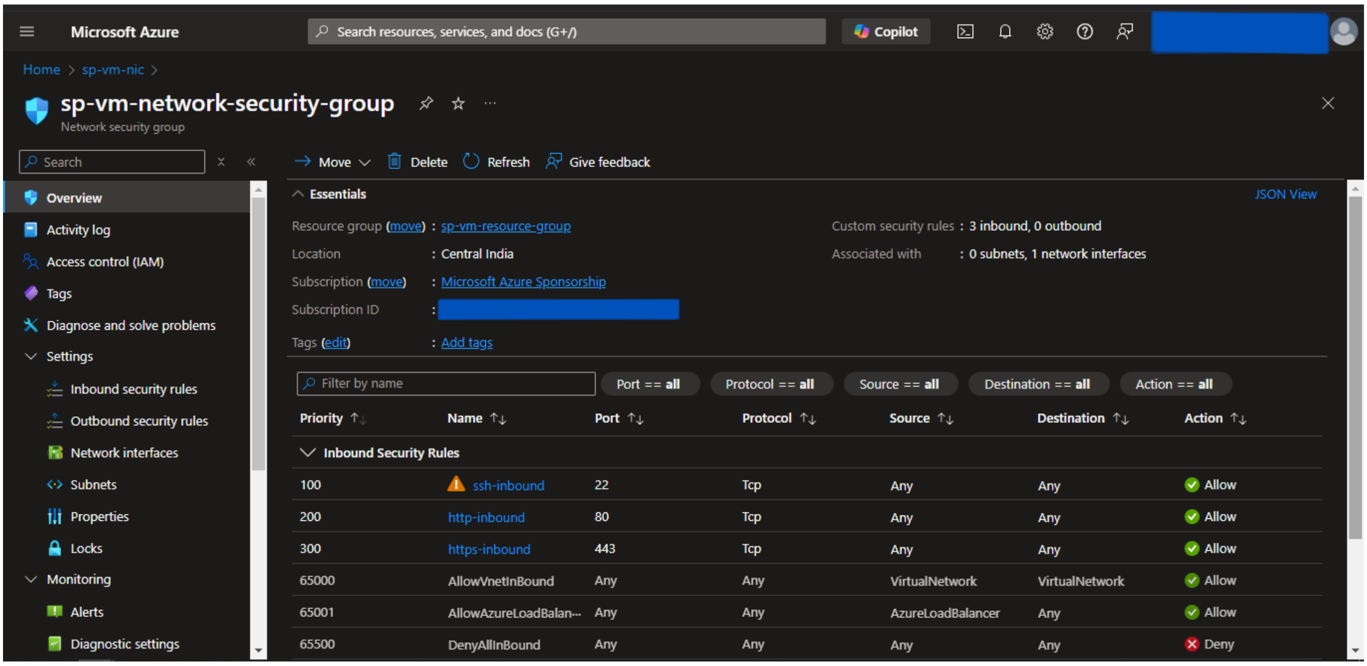
Tags (edit) : Add tags

Enable Azure resources to communicate with each other

A network interface allows Azure resources to communicate with the internet, and on-premises resources. Learn more

---

Network Security Group Image



[redacted]

sp-vm-network-security-group
Network security group

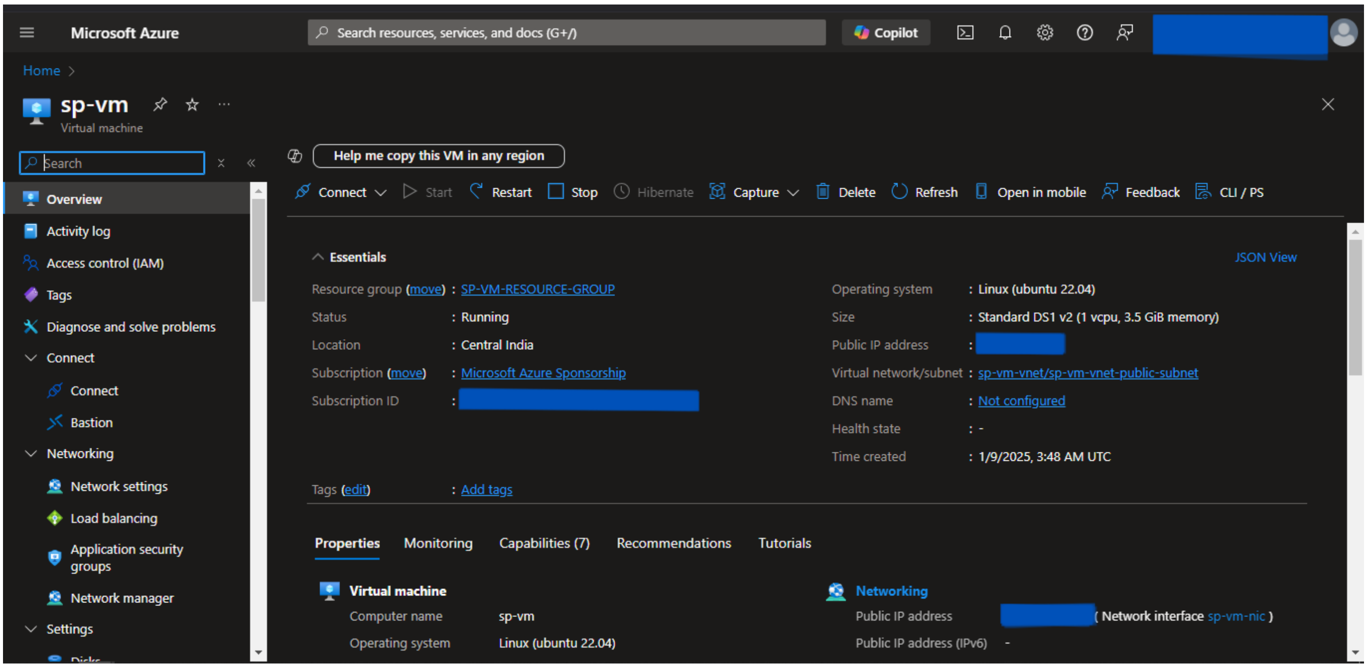
Essentials

Resource group (move) : sp-vm-resource-group
Location : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Tags (edit) : Add tags
Custom security rules : 3 inbound, 0 outbound
Associated with : 0 subnets, 1 network interfaces

Inbound Security Rules

Priority	Name	Port	Protocol	Source	Destination	Action
100	⚠ ssh-inbound	22	Tcp	Any	Any	✓ Allow
200	http-inbound	80	Tcp	Any	Any	✓ Allow
300	https-inbound	443	Tcp	Any	Any	✓ Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	✓ Allow
65001	AllowAzureLoadBalan...	Any	Any	AzureLoadBalancer	Any	✓ Allow
65500	DenyAllInBound	Any	Any	Any	Any	✗ Deny

Virtual Machine Image



[redacted]

sp-vm
Virtual machine

Essentials

Resource group (move) : SP-VM-RESOURCE-GROUP
Status : Running
Location : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Operating system : Linux (ubuntu 22.04)
Size : Standard DS1 v2 (1 vcpu, 3.5 GiB memory)
Public IP address : [redacted]
Virtual network/subnet : sp-vm-vnet/sp-vm-vnet-public-subnet
DNS name : Not configured
Health state : -
Time created : 1/9/2025, 3:48 AM UTC
Tags (edit) : Add tags

Virtual machine
Computer name sp-vm
Operating system Linux (ubuntu 22.04)

Networking
Public IP address [redacted] (Network interface sp-vm-nic)
Public IP address (IPv6) -

SSH Into Azure VM

Now we will SSH into the Azure VM and configure it for Nginx container deployment.

Steps

1. Open the Powershell Window.
2. Run the following command to SSH into Azure VM and substitute the *<admin-username>* with the value provided in *locals.tf* file under *<virtual-machine-properties>* section and *<vm-public-ip>* with the Azure VM Public IP received from `terraform output` command:

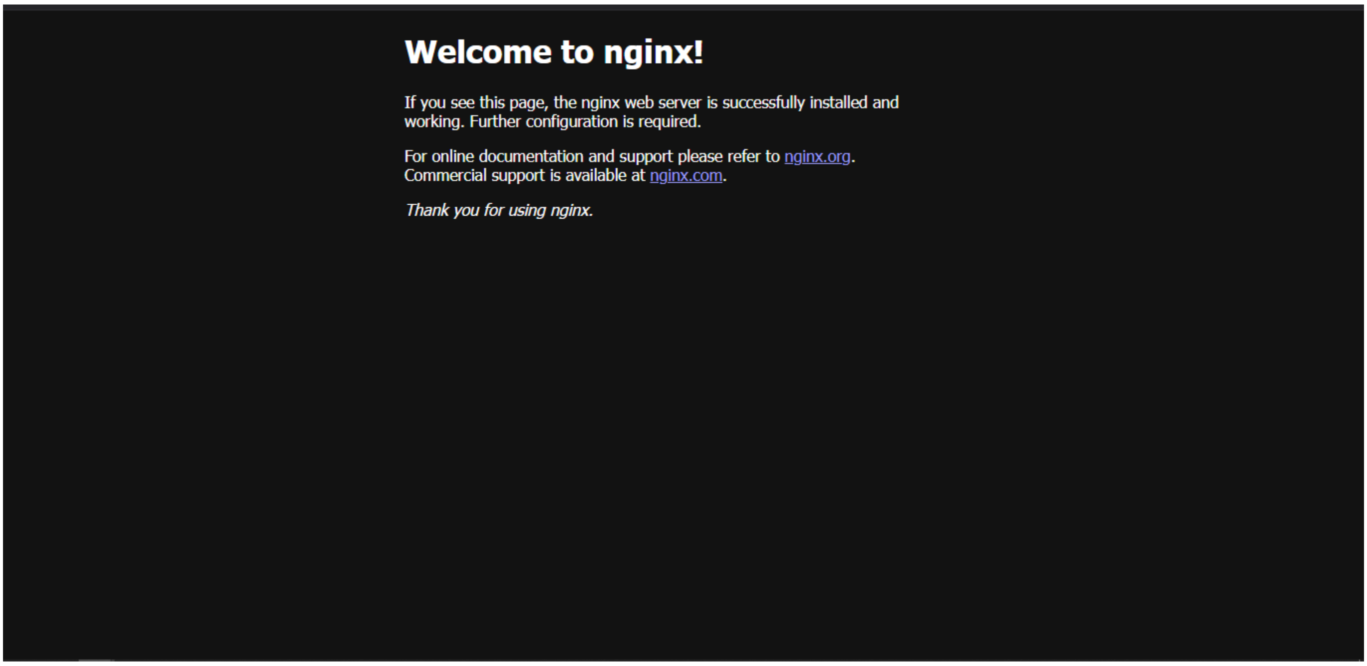
```
ssh -o StrictHostKeyChecking=no <admin-username>@<vm-public-ip>
```

3. It will promt for password, enter the *<admin-password>* provided in the *locals.tf* file under *<virtual-machine-properties>* section.
4. Once you enter the server, run the following commands to install the necessary dependencies for deployment and run the nginx container:

```
sudo apt update
sudo apt install -y docker.io
sudo docker run -d -p 80:80 nginx
```

9. Try accessing it on the browser using *<vm-public-ip>* received from `terraform output` command.

Nginx Image



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

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 **virtual-machine-terraform** directory using the `cd` command.
2. Run `terraform destroy` & if prompted, type `yes`.
3. Infrastructure will be destroyed.

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.

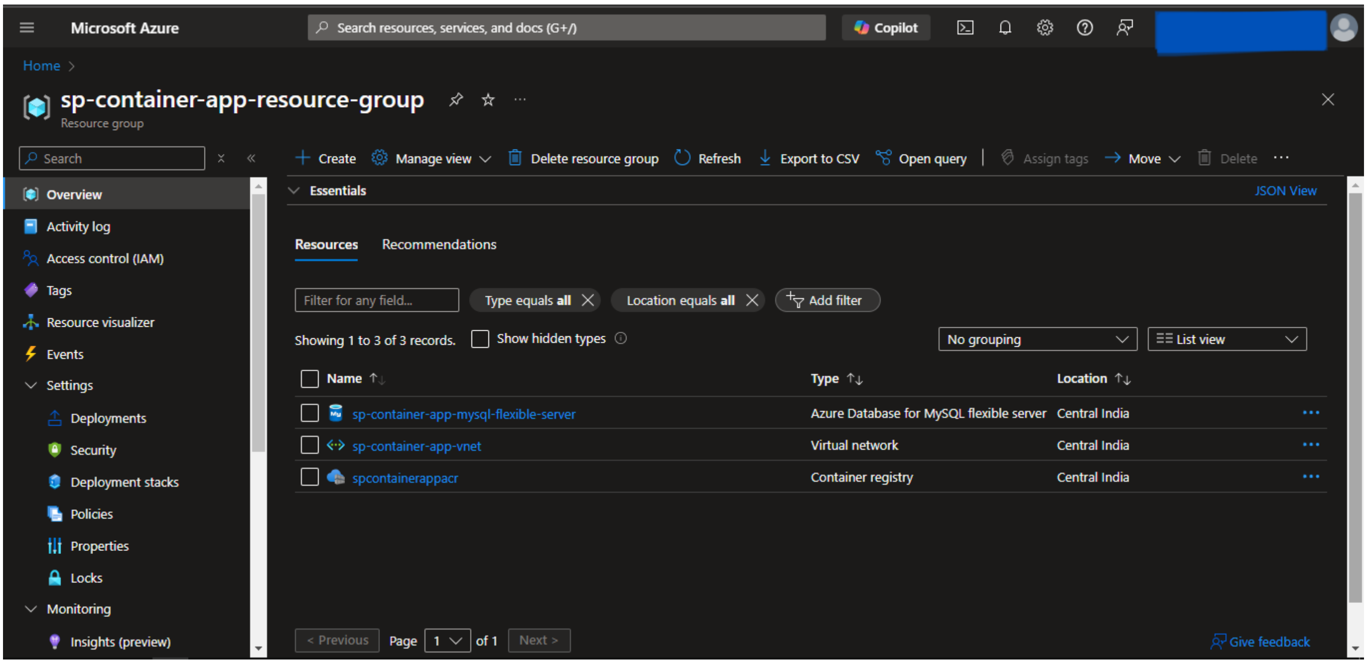
> Also first provision the ACR, push the Docker Image, and then provision the Container App. To do that, comment out the **container-app** module and follow the further steps.

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.

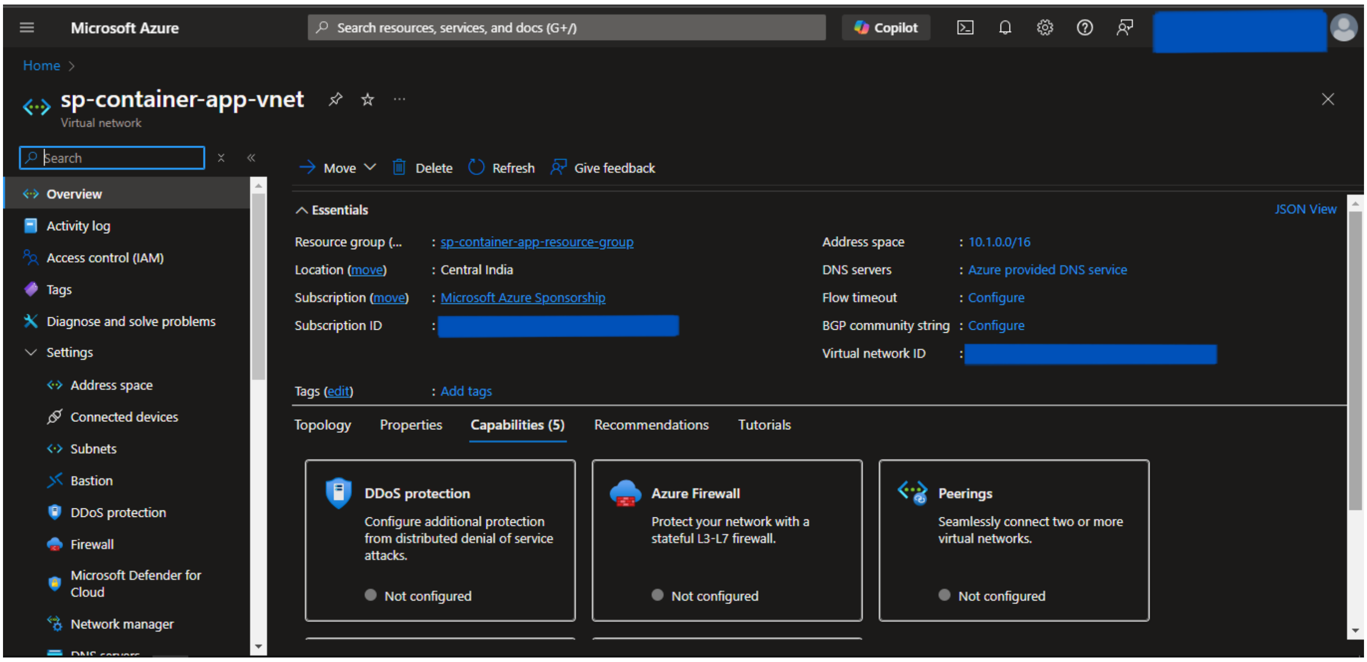
Screenshots of Provisioned Infrastructure

Resource Group Image



Name	Type	Location
sp-container-app-mysql-flexible-server	Azure Database for MySQL flexible server	Central India
sp-container-app-vnet	Virtual network	Central India
spcontainerappacr	Container registry	Central India

Virtual Network Image



[redacted]

sp-container-app-vnet
Virtual network

Essentials

Resource group (... : sp-container-app-resource-group
Location (move) : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Tags (edit) : Add tags

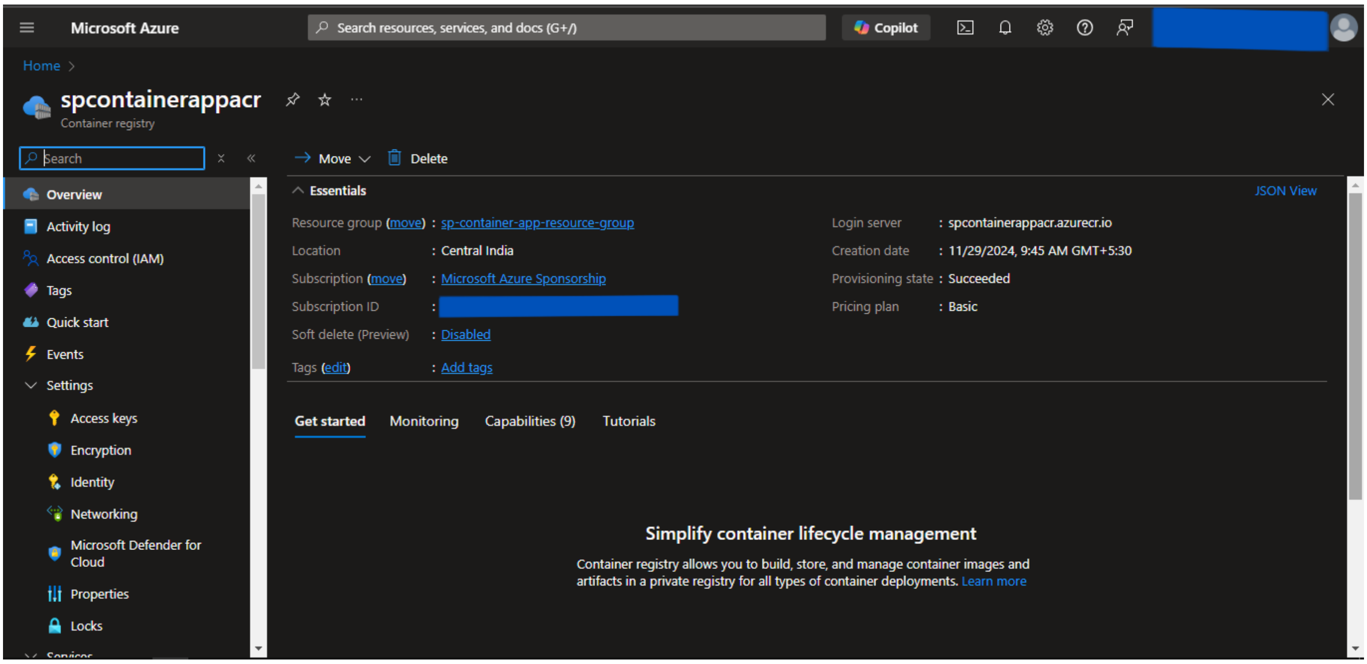
Address space : 10.1.0.0/16
DNS servers : Azure provided DNS service
Flow timeout : Configure
BGP community string : Configure
Virtual network ID : [redacted]

DDoS protection
Configure additional protection from distributed denial of service attacks.
● Not configured

Azure Firewall
Protect your network with a stateful L3-L7 firewall.
● Not configured

Peerings
Seamlessly connect two or more virtual networks.
● Not configured

ACR Image



[redacted]

spcontainerappacr
Container registry

Essentials

Resource group (move) : sp-container-app-resource-group
Location : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Soft delete (Preview) : Disabled
Tags (edit) : Add tags

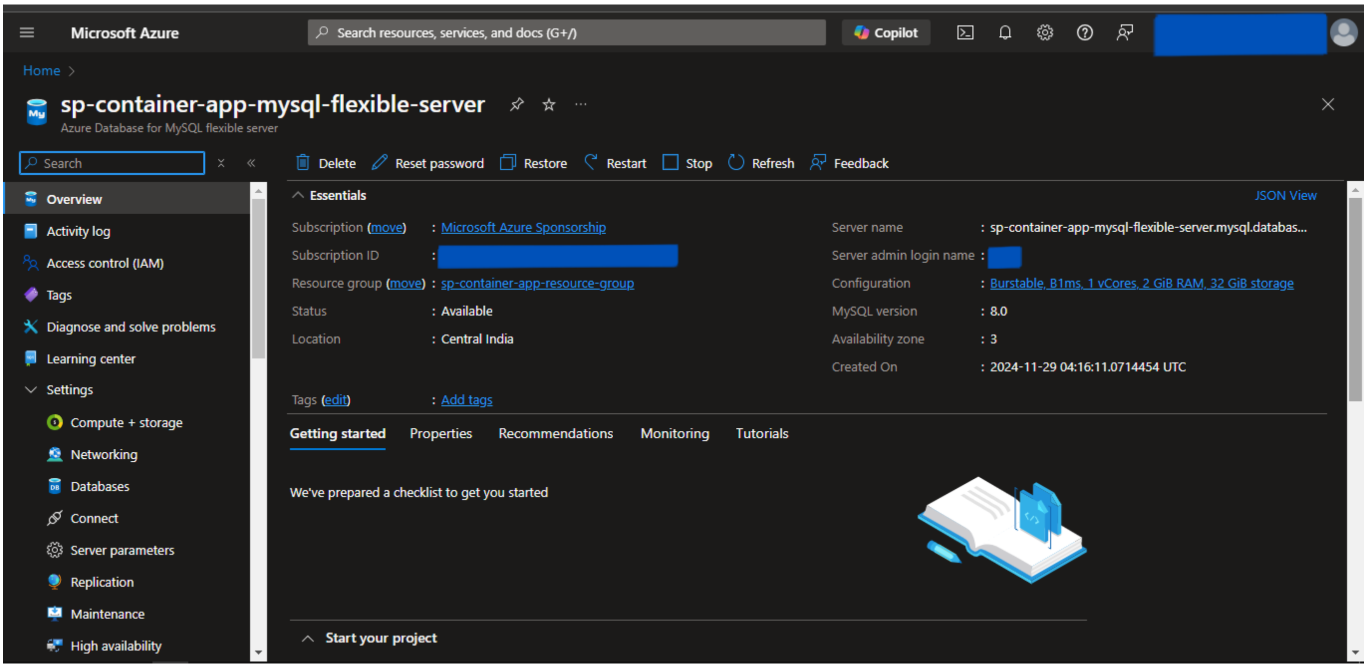
Login server : spcontainerappacr.azurecr.io
Creation date : 11/29/2024, 9:45 AM GMT+5:30
Provisioning state : Succeeded
Pricing plan : Basic

Simplify container lifecycle management

Container registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Learn more

---

MySQL Flexible Server Image



[redacted]

sp-container-app-mysql-flexible-server

Azure Database for MySQL flexible server

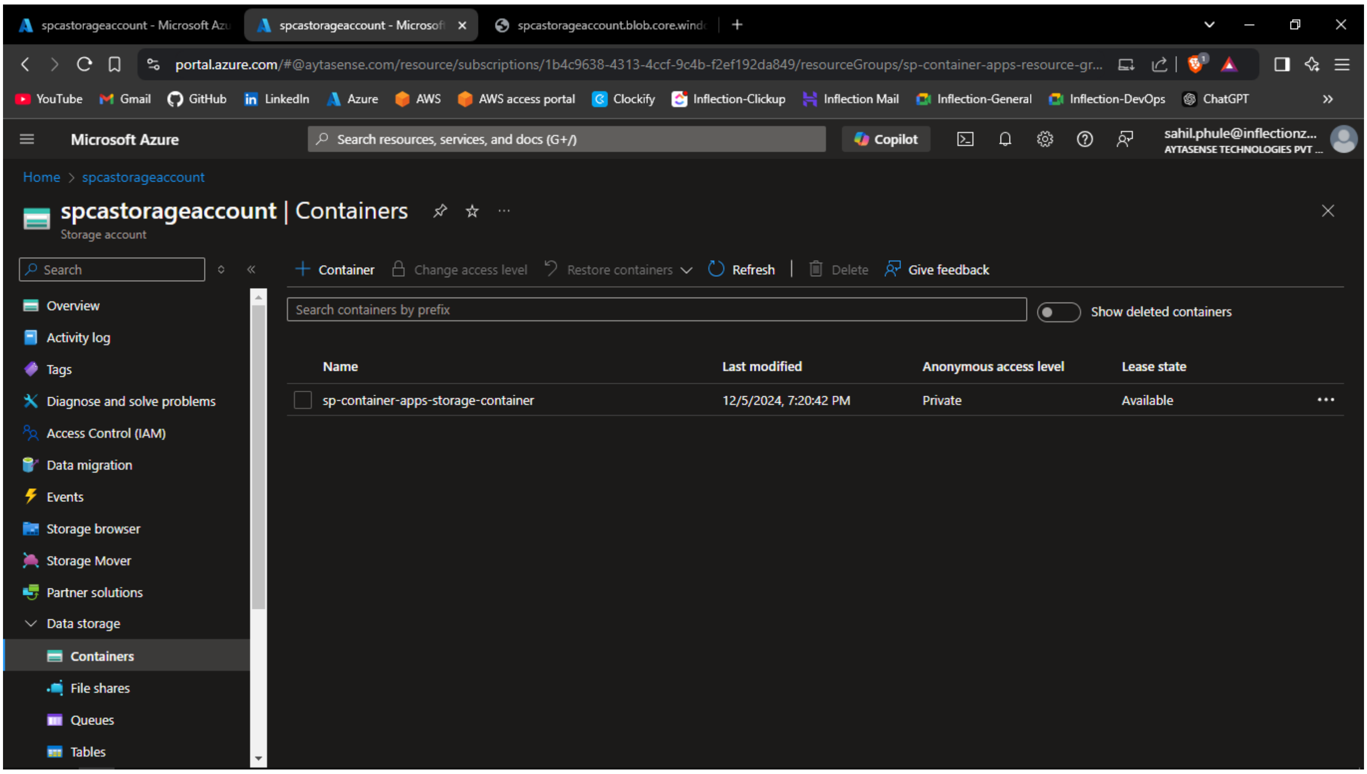
Essentials

Subscription (move)	: Microsoft Azure Sponsorship
Subscription ID	: [redacted]
Resource group (move)	: sp-container-app-resource-group
Status	: Available
Location	: Central India
Tags (edit)	: Add tags

Server name	: sp-container-app-mysql-flexible-server.mysql.databas...
Server admin login name	: [redacted]
Configuration	: Burstable, B1ms, 1 vCores, 2 GiB RAM, 32 GiB storage
MySQL version	: 8.0
Availability zone	: 3
Created On	: 2024-11-29 04:16:11.0714454 UTC

We've prepared a checklist to get you started

Storage Account Container Image



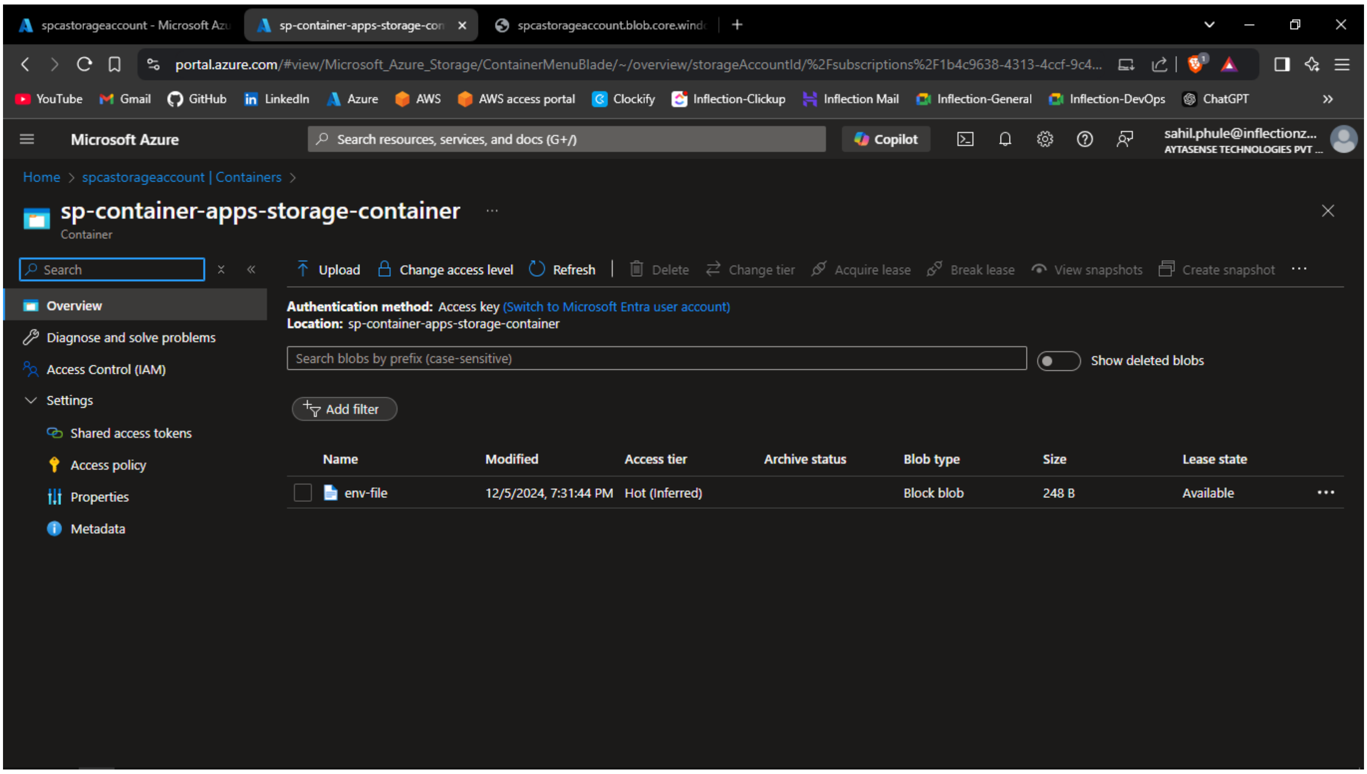
portal.azure.com/#@aytasense.com/resource/subscriptions/1b4c9638-4313-4ccf-9c4b-f2ef192da849/resourceGroups/sp-container-apps-resource-gr...

spcastorageaccount | Containers

Storage account

	Name	Last modified	Anonymous access level	Lease state
□	sp-container-apps-storage-container	12/5/2024, 7:20:42 PM	Private	Available

Storage Account Container Env File


spcastorageaccount - Microsoft Azu
sp-container-apps-storage-con
spcastorageaccount.blob.core.wind
portal.azure.com/#view/Microsoft_Azure_Storage/ContainerMenuBlade/~/overview/storageAccountId/%2Fsubscriptions%2F1b4c9638-4313-4ccf-9c4...
YouTube
Gmail
GitHub
LinkedIn
Azure
AWS
AWS access portal
Clockify
Inflection-Clickup
Inflection Mail
Inflection-General
Inflection-DevOps
ChatGPT
Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
sahil.phule@inflectionz...
AYTASENSE TECHNOLOGIES PVT ...
Home > spcastorageaccount | Containers >
sp-container-apps-storage-container
Container
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Shared access tokens
Access policy
Properties
Metadata
Upload
Change access level
Refresh
Delete
Change tier
Acquire lease
Break lease
View snapshots
Create snapshot
Search blobs by prefix (case-sensitive)
Show deleted blobs
Add filter

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: sp-container-apps-storage-container

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
env-file	12/5/2024, 7:31:44 PM	Hot (Inferred)		Block blob	248 B	Available

---

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.

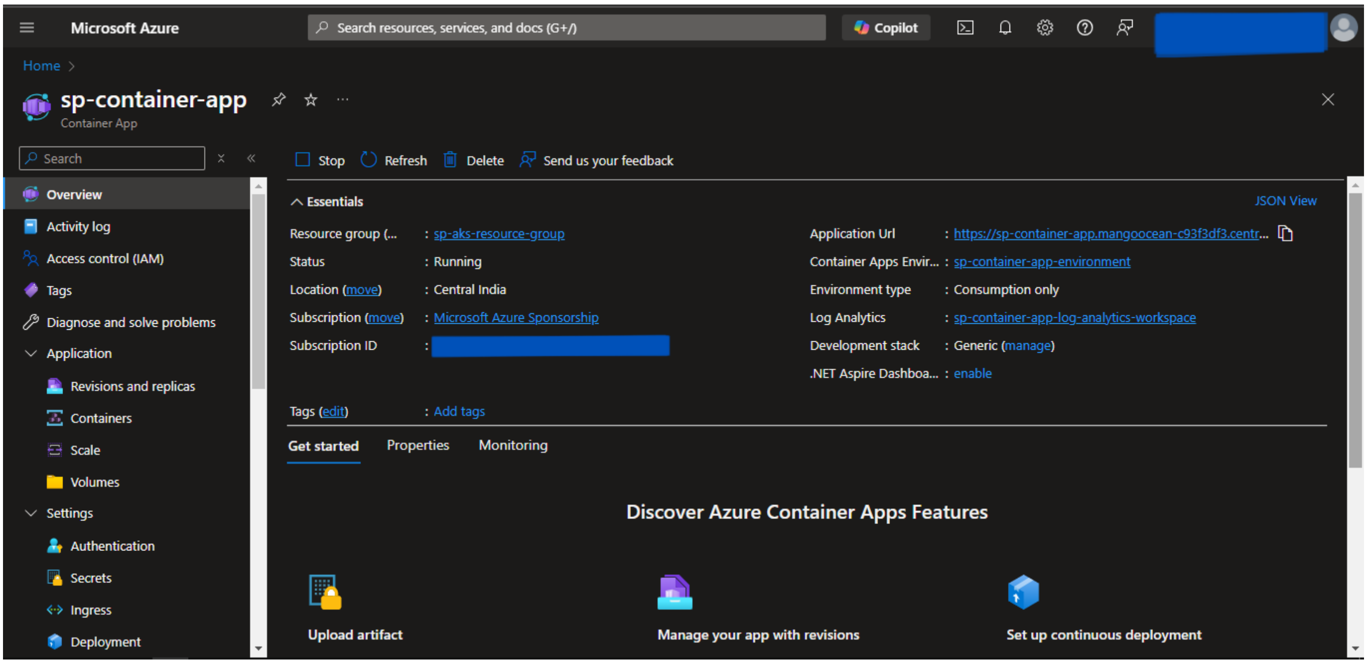
Provisioning the Container App

> 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



Resource group (... : sp-aks-resource-group
Status : Running
Location (move) : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Application Url : https://sp-container-app.mangoocean-c93f3df3.centr...
Container Apps Envir... : sp-container-app-environment
Environment type : Consumption only
Log Analytics : sp-container-app-log-analytics-workspace
Development stack : Generic (manage)
.NET Aspire Dashboa... : enable
Tags (edit) : Add tags

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.

AKS Provisioning using Terraform

- We will provision the AKS using Terraform as an Infrastructure as Code.
- We will deploy it in a custom Virtual Network for isolation.
- We will connect the AKS to ACR for Docker Image.
- We will also deploy MySQL Flexible to store the relational data and connect it to AKS.

Prerequisites

---

1. Azure Account with Subscription.
2. Terraform installed.
3. Kubectl 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 **aks-terraform** directory.
2. The folder structure for the above-created directory is as follows:

```
aks-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.aks
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
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.aks-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.

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.

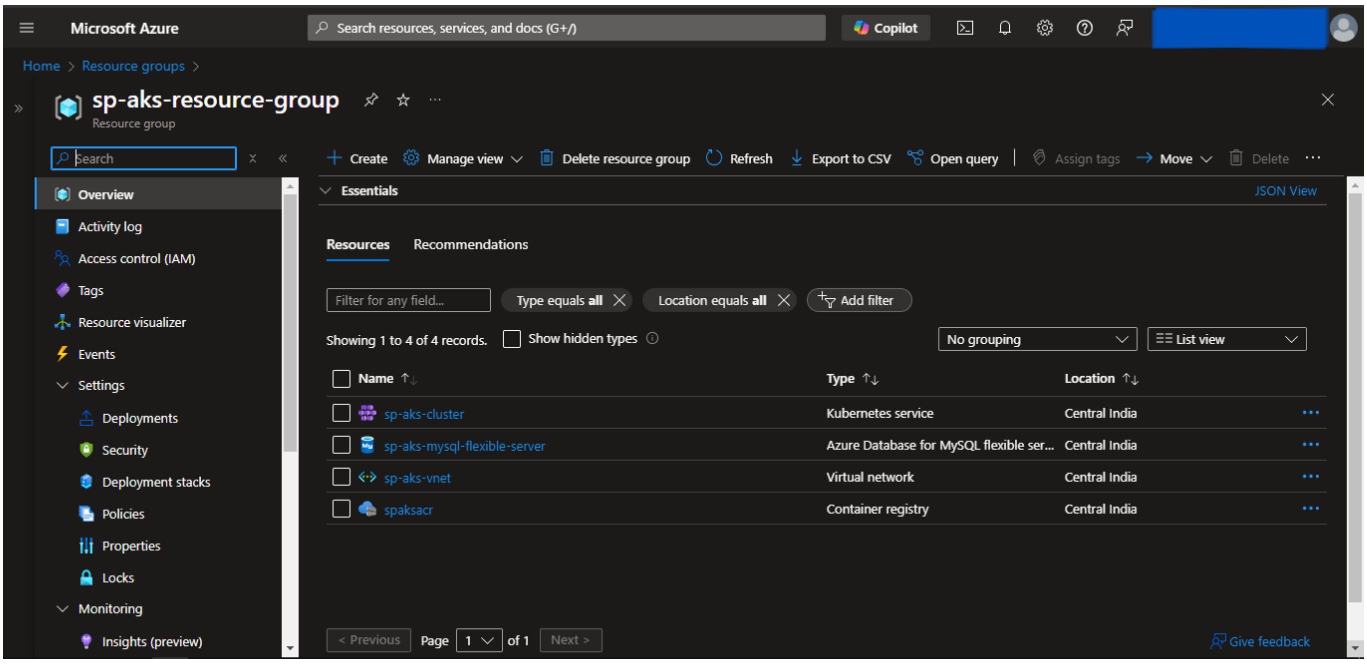
> Also first provision the ACR, push the Docker Image, and then provision the Container App. To do that, comment out the **aks** module and follow the further steps.

Steps:

1. Open the PowerShell.
2. Change the directory to the above-created **aks-terraform** directory using `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. Run the `terraform output` command to get the values of defined variables in *outputs.tf* file.
9. Head to the Azure Console, and verify the created resources.

Screenshots of Provisioned Infrastructure

Resource Group Image

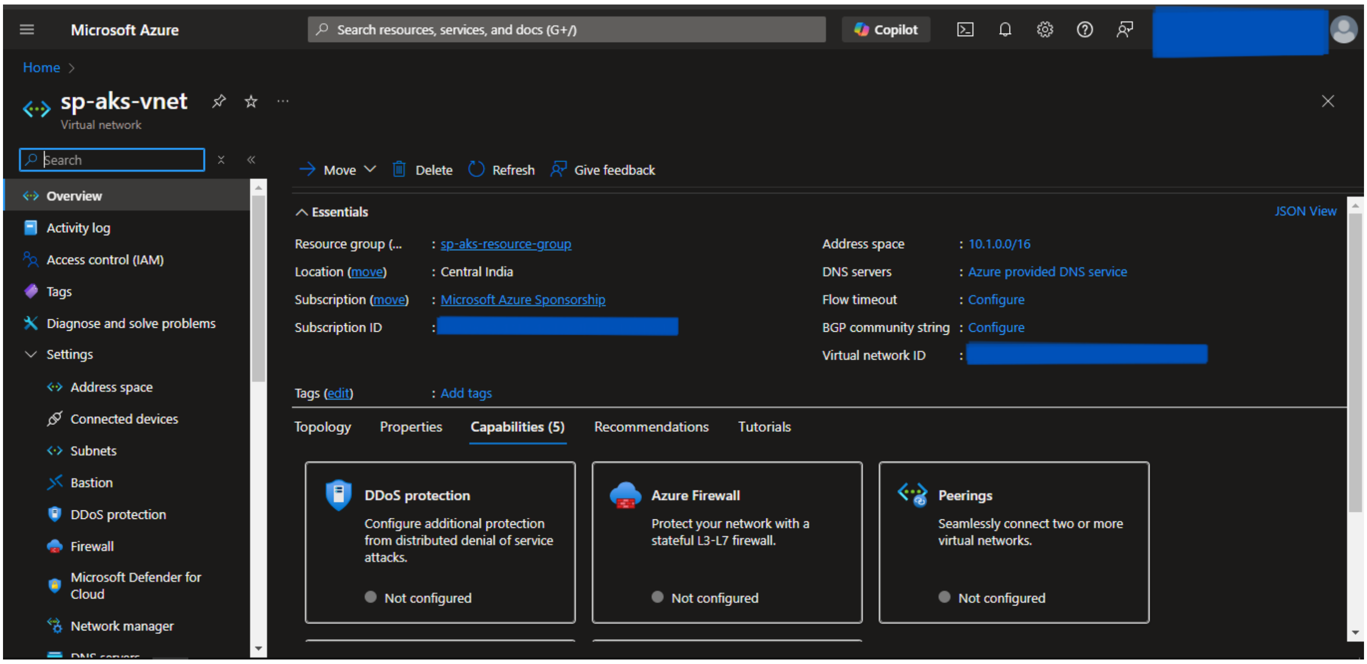


sp-aks-resource-group

Showing 1 to 4 of 4 records. □ Show hidden types

Name	Type	Location
sp-aks-cluster	Kubernetes service	Central India
sp-aks-mysql-flexible-server	Azure Database for MySQL flexible ser...	Central India
sp-aks-vnet	Virtual network	Central India
spaksacr	Container registry	Central India

Virtual Network Image



sp-aks-vnet
Virtual network

Essentials

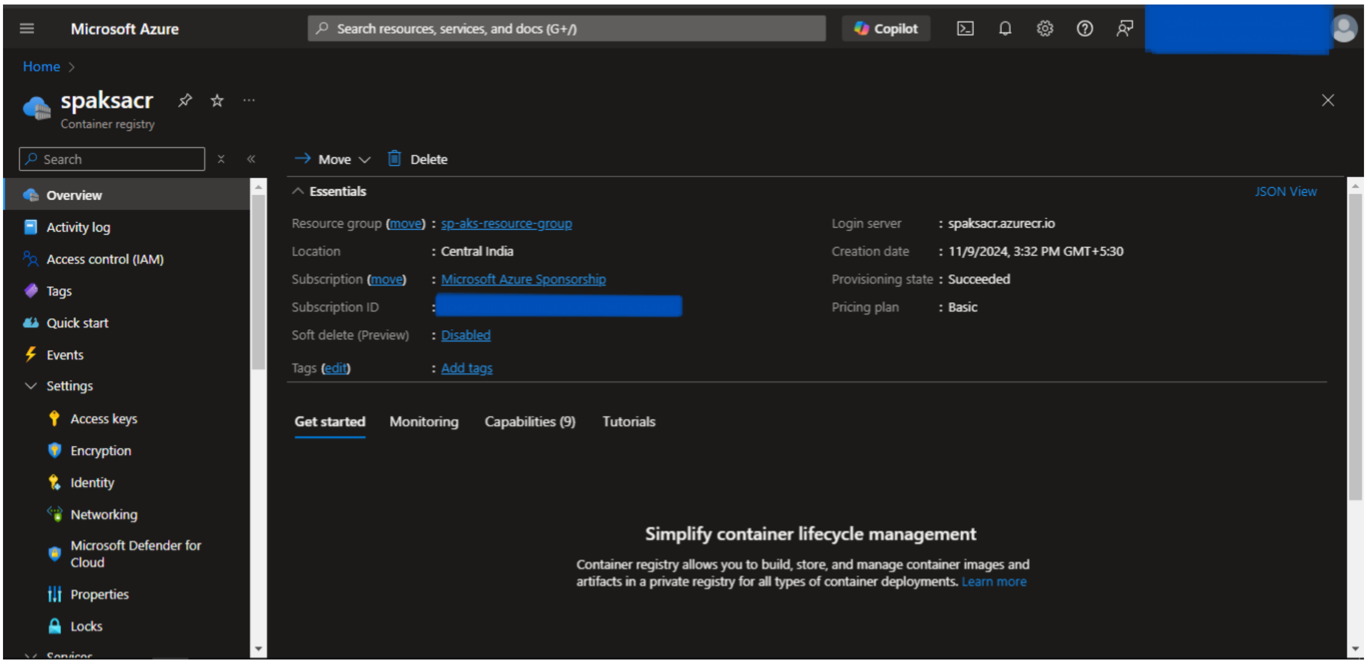
Resource group (... : sp-aks-resource-group
Location (move) : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Address space : 10.1.0.0/16
DNS servers : Azure provided DNS service
Flow timeout : Configure
BGP community string : Configure
Virtual network ID : [redacted]
Tags (edit) : Add tags

DDoS protection
Configure additional protection from distributed denial of service attacks.
● Not configured

Azure Firewall
Protect your network with a stateful L3-L7 firewall.
● Not configured

Peerings
Seamlessly connect two or more virtual networks.
● Not configured

ACR Image



spaksacr
Container registry

Essentials

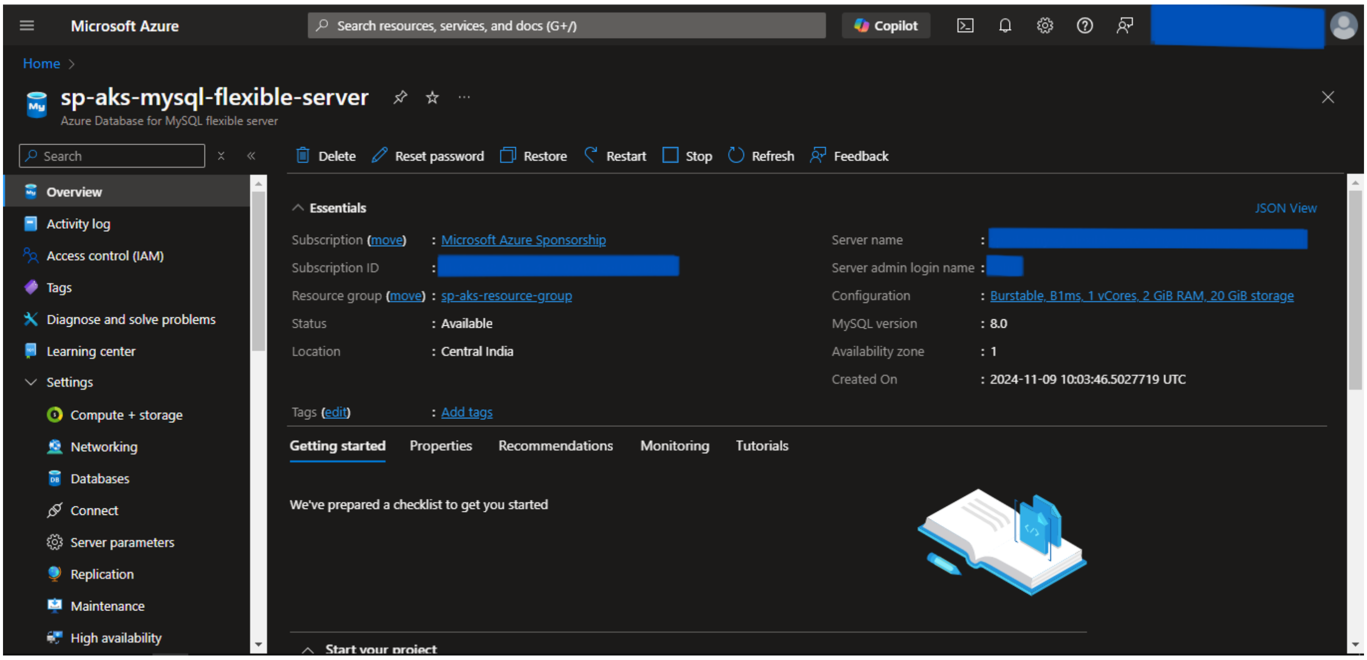
Resource group (move) : sp-aks-resource-group
Location : Central India
Subscription (move) : Microsoft Azure Sponsorship
Subscription ID : [redacted]
Soft delete (Preview) : Disabled
Tags (edit) : Add tags
Login server : spaksacr.azurecr.io
Creation date : 11/9/2024, 3:32 PM GMT+5:30
Provisioning state : Succeeded
Pricing plan : Basic

Simplify container lifecycle management

Container registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Learn more

---

MySQL Flexible Server Image



[redacted]

sp-aks-mysql-flexible-server

Azure Database for MySQL flexible server

Essentials

Subscription (move) : Microsoft Azure Sponsorship

Subscription ID : [redacted]

Resource group (move) : sp-aks-resource-group

Status : Available

Location : Central India

Server name : [redacted]

Server admin login name : [redacted]

Configuration : Burstable, B1ms, 1 vCores, 2 GiB RAM, 20 GiB storage

MySQL version : 8.0

Availability zone : 1

Created On : 2024-11-09 10:03:46.5027719 UTC

Tags (edit) : Add tags

We've prepared a checklist to get you started

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.

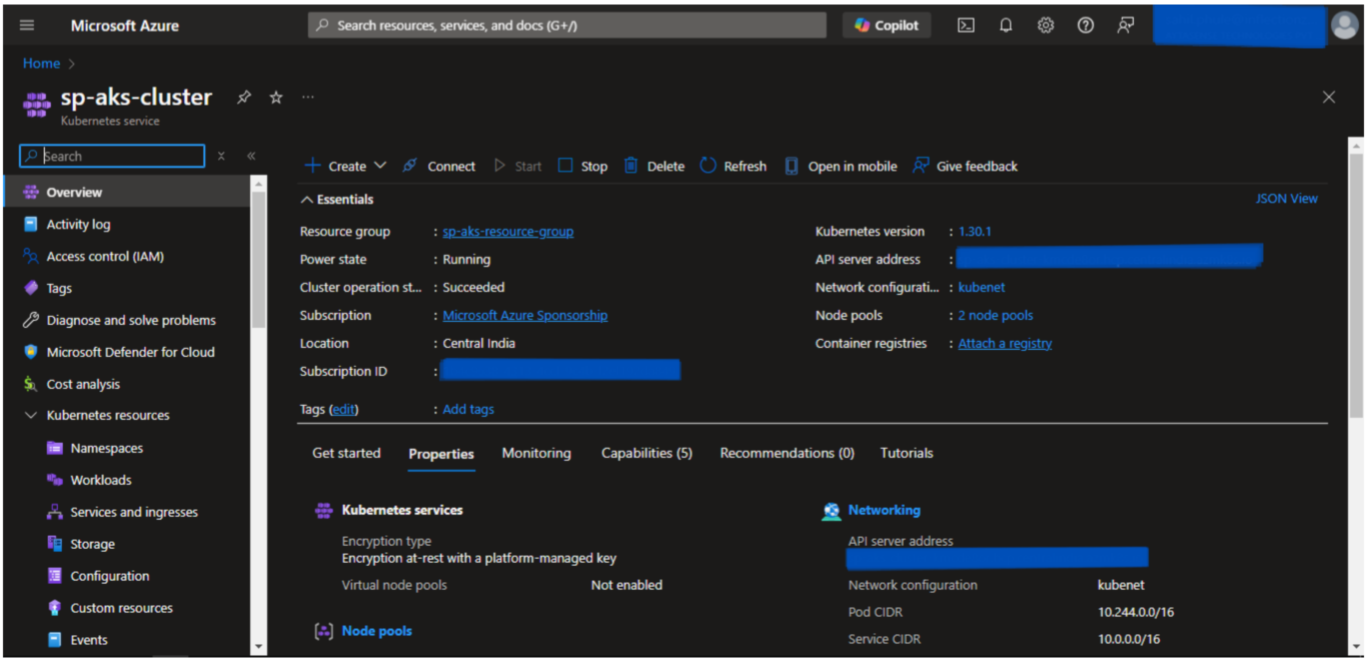
Provisioning the AKS

> Uncomment the **aks** module that we commented on earlier and follow the further steps.

Steps:

1. Open the Powershell Window.
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.

AKS Image



Resource group : sp-aks-resource-group
Power state : Running
Cluster operation st... : Succeeded
Subscription : Microsoft Azure Sponsorship
Location : Central India
Subscription ID : [redacted]
Tags (edit) : Add tags

Kubernetes version : 1.30.1
API server address : [redacted]
Network configurati... : kubenet
Node pools : 2 node pools
Container registries : Attach a registry

Kubernetes services
Encryption type
Encryption at-rest with a platform-managed key
Virtual node pools Not enabled

Node pools

Networking
API server address
[redacted]
Network configuration kubenet
Pod CIDR 10.244.0.0/16
Service CIDR 10.0.0.0/16

Connect to the AKS Cluster from Powershell

Steps

1. Open a new Powershell window.
2. Run the following commands to configure local kubectl with aks cluster:

```
az login
az account set --subscription <subscription-id>
az aks get-credentials --resource-group <resource-group-name> --name <cluster-name> --overwrite-existing
```

> Substitute <*subscription-id*> which can be found by running `az account list` in the *id* field. Also, substitute <*resource-group-name*> and <*cluster-name*> with the values defined in the above-created *locals.tf* file.

3. Now apply the Kubernetes manifest files of the application using the following command:

```
kubectl apply -f <file-path>
```

> Substitute with the Kubernetes manifest file path.

4. To list them all, run `kubectl get all`.
5. If a Load Balancer type Service is present then try accessing the External IP of that service in the browser.

Destroy the provisioned infrastructure

Lastly, we will destroy the above-created resources.

Steps

1. Firstly, delete all the Kubernetes Deployments using:
    - `kubectl delete -f "file-path"`
      Substitute *file-path* with the Kubernetes manifest file path.
2. To destroy infrastructure, change the directory to the above-created **aks-terraform** directory using the `cd` command.
3. Run `terraform destroy` & if prompted, type `yes`.
4. Infrastructure will be destroyed.