Devops Project Report
On
Deploying a 2-tier Application on AWS Cloud Using Terraform and and Docker Containers

Submitted by

Name :Biswa Prakash Rout
Reg no :2241018146
Branch :CSE
Sl no :31

1. INTRODUCTION

In the current digital era, businesses demand fast, repeatable, and error-free deployment of their applications. Manual server setup and configuration often lead to inefficiencies and scalability issues. This project demonstrates the deployment of a 2-tier web application on AWS using Infrastructure as Code (IaC) with Terraform and containerization with Docker. By automating infrastructure provisioning and packaging applications into containers, this project provides a consistent and reliable deployment environment while reducing operational overhead.

2. PROBLEM STATEMENT

Using a Bash Scripting, create 3 stages. Each of these stages has a specific

job. The stages are named as such:

1. Create_Infra

Create a 2-tier application (preferably on git) using any language andtools, run and test the application.The application must have a frontend and a database connected to it inthe backend. It must allow the user to enter some details in the frontendand store the same in a row in the database.

2. Deploy_Apps

Using terraform provisioners execute the scripts in respective systems:frontend.sh: Installs and Configures docker in the FRONTEND instance and runsthe containerized frontend in it using the pull request command in previous slide.backend.sh: Installs and Configures docker in BACKEND instance and runs thecontainerized backend in it using the pull request command in previous slide.Use remote-exec provisioner to find out if docker has been installed and the application is running in the local system.

Stage 3: Test_Solution

Using terraform output save the public DNS or Public IP of the FRONTEND and display it as the stage is executed.Using terraform outputs and variables display the exact address with port number for the frontend form application.Curl the public DNS or IP to check if the frontend containerized application is working.

Manual Testing

Manually test if the application is running correctly. In the frontend enter some details for an user (Capture a screenshot of it)Now dive into the database and check if the line has been added to the database in the BACKEND instance or not (Capture a screenshot of it)

3. PROJECT OVERVIEW

This project focuses on building a 2-tier application deployment architecture on AWS. The architecture consists of:

- **Frontend Layer:** A web interface running in a Docker container that interacts with users.
- **Backend Layer:** A containerized API that connects to a database hosted on MongoDB Atlas for data management.

Terraform is used to provision AWS resources like VPC, subnets, security groups, and EC2 instances, while Docker ensures the applications run in isolated, portable environments. By combining Terraform and Docker, the project delivers an automated, cost-effective, and scalable solution suitable for modern cloud-based application deployments.

Devops Project

1.Terraform files

main.tf

```
provider "aws" {
  region = “eu-north-1a”
}

# VPC

resource "aws_vpc" "my_vpc" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = "devops2tier-vpc"
  }
}


# Public Subnet

resource "aws_subnet" "public_subnet" {
  vpc_id                  = aws_vpc.my_vpc.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "eu-north-1a"

  tags = {
    Name = "public-subnet"
  }
}


# Internet Gateway

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.my_vpc.id
  tags = {
    Name = "devops2tier-igw"
```

```
  }
}


# Route Table for Public Subnet

resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.my_vpc.id
  tags = {
    Name = "public-rt"
  }
}

resource "aws_route" "public_internet_access" {
  route_table_id         = aws_route_table.public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.igw.id
}

resource "aws_route_table_association" "public_assoc" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
}


# Security Groups

resource "aws_security_group" "frontend_sg" {
  name        = "frontend-sg"
  description = "Allow HTTP and SSH"
  vpc_id      = aws_vpc.my_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "frontend-sg"
  }
}

resource "aws_security_group" "backend_sg" {
  name        = "backend-sg"
  description = "Allow backend API access"
  vpc_id      = aws_vpc.my_vpc.id

  ingress {
    from_port   = 5000
    to_port     = 5000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "backend-sg"
  }
}


# Frontend EC2

resource "aws_instance" "frontend" {
  ami                    = var.ami_id
  instance_type          = "t3.micro"
```

```
  subnet_id              = aws_subnet.public_subnet.id
  security_groups        = [aws_security_group.frontend_sg.id]
  key_name               = var.key_name

  user_data = <<-EOF
#!/bin/bash
apt update -y
apt install -y docker.io
systemctl start docker

# Create index.html
cat <<EOT > /home/ubuntu/index.html
<!DOCTYPE html>
<html>
<head>
    <title>DevOps Login</title>
    <style>
        body { background-color: #282c34; color: #61dafb; font-family: Arial, sans-serif; text-align:
center; }
        form { margin-top: 30px; background: #333; padding: 25px; border-radius: 8px; display:
inline-block; text-align: left; width: 300px; }
        .form-header { font-size: 22px; font-weight: bold; text-align: center; margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; }
        input[type="email"], input[type="password"] { width: 100%; padding: 8px; border: 1px solid
#61dafb; border-radius: 4px; margin-bottom: 15px; background: #222; color: #61dafb; }
        input[type="submit"] { width: 100%; background: #61dafb; border: none; padding: 10px;
cursor: pointer; color: #000; font-weight: bold; margin-top: 5px; }
        .forgot { text-align: center; margin-top: 10px; font-size: 12px; color: #61dafb; cursor:
pointer; }
        .success { margin-top: 15px; text-align: center; color: limegreen; font-weight: bold; }
    </style>
</head>
<body>
    <h1>Welcome here ! Biswa this side</h1>
    <form id="loginForm">
        <div class="form-header">Login</div>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <input type="submit" value="Submit">
        <div class="forgot">Forgot Password?</div>
        <div id="result" class="success"></div>
    </form>
```

```
    <script>
      document.getElementById("loginForm").addEventListener("submit", async function(e) {
        e.preventDefault();
        const email = document.getElementById("email").value;
        const password = document.getElementById("password").value;
        const response = await fetch("http://${aws_instance.backend.public_ip}:5000/save", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ email, password })
        });
        const result = await response.json();
        document.getElementById("result").innerText = result.message;
      });
    </script>
</body>
</html>
EOT

docker run -d -p 80:80 --name frontend -v
/home/ubuntu/index.html:/usr/share/nginx/html/index.html nginx
EOF
}


# Backend EC2

resource "aws_instance" "backend" {
  ami                         = var.ami_id
  instance_type               = "t3.micro"
  subnet_id                   = aws_subnet.public_subnet.id
  associate_public_ip_address = true
  vpc_security_group_ids      = [aws_security_group.backend_sg.id]
  key_name                    = var.key_name

  user_data = <<-EOF
#!/bin/bash
exec > >(tee /var/log/user_data.log|logger -t user-data ) 2>&1
set -x

apt update -y
apt install -y docker.io
systemctl start docker

mkdir -p /home/ubuntu/backend
```

```
cd /home/ubuntu/backend

# Create Flask backend app
cat <<EOT > app.py
from flask import Flask, request, jsonify
from flask_cors import CORS
from pymongo import MongoClient

app = Flask(__name__)
CORS(app)

print("Starting Flask app and connecting to MongoDB Atlas...")
client =
MongoClient("mongodb+srv://dbuser:dbpass123@cluster0.i3gcqrs.mongodb.net/devopsdb?retr
yWrites=true&w=majority")
db = client["devopsdb"]
users = db["users"]

@app.route('/save', methods=['POST'])
def save():
    data = request.json
    email = data.get("email")
    password = data.get("password")

    if not email or not password:
        return jsonify({"message": "Email and Password required"}), 400

    users.insert_one({"email": email, "password": password})
    print(f"Inserted user: {email}")
    return jsonify({"message": "Login data stored successfully!"})

if __name__ == '__main__':
    print("Flask backend running on port 5000")
    app.run(host='0.0.0.0', port=5000)
EOT

# Create Dockerfile
cat <<EOT > Dockerfile
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install flask flask-cors pymongo
CMD ["python", "app.py"]
EOT
```

```
docker build -t biswaa18/backend:1.0 .
docker run -d --name backend -p 5000:5000 biswaa18/backend:1.0

# Log container status
docker ps
docker logs -f backend > /var/log/backend.log 2>&1 &
EOF
}
```

Terraform.tfvars

```
key_name = "devops-key-1754072235"
private_key_path = "~/.ssh/your-key.pem"
```

Terraform-outputs.txt

```
output "frontend_public_ip" {
  value = aws_instance.frontend.public_ip
}

output "frontend_url" {
  value = "http://${aws_instance.frontend.public_ip}:80"
}
```

Variables.tf

```
variable "aws_region" {
  default = "eu-north-1"
}
variable "ami_id" {
  description = "Ubuntu AMI ID"
  default     = "ami-07a0715df72e58928" # Ubuntu 22.04 in eu-north-1
}
variable "key_name" {
  description = "Name of your AWS key pair"
}

variable "private_key_path" {
  description = "Path to your private key file"
}
```

2. Containerized Application Link (provide your link from dockerhub)

https://hub.docker.com/repository/docker/biswaa18/frontend/general

https://hub.docker.com/repository/docker/biswaa18/backend/general

3.Frontend.sh

```
#!/bin/bash
sudo apt update -y
sudo apt install docker.io -y
sudo systemctl enable docker
sudo systemctl start docker
sudo usermod -aG docker ubuntu

Sudo docker run -d -p 80:80 biswaa18/frontend-app:latest
```

4. Backend.sh

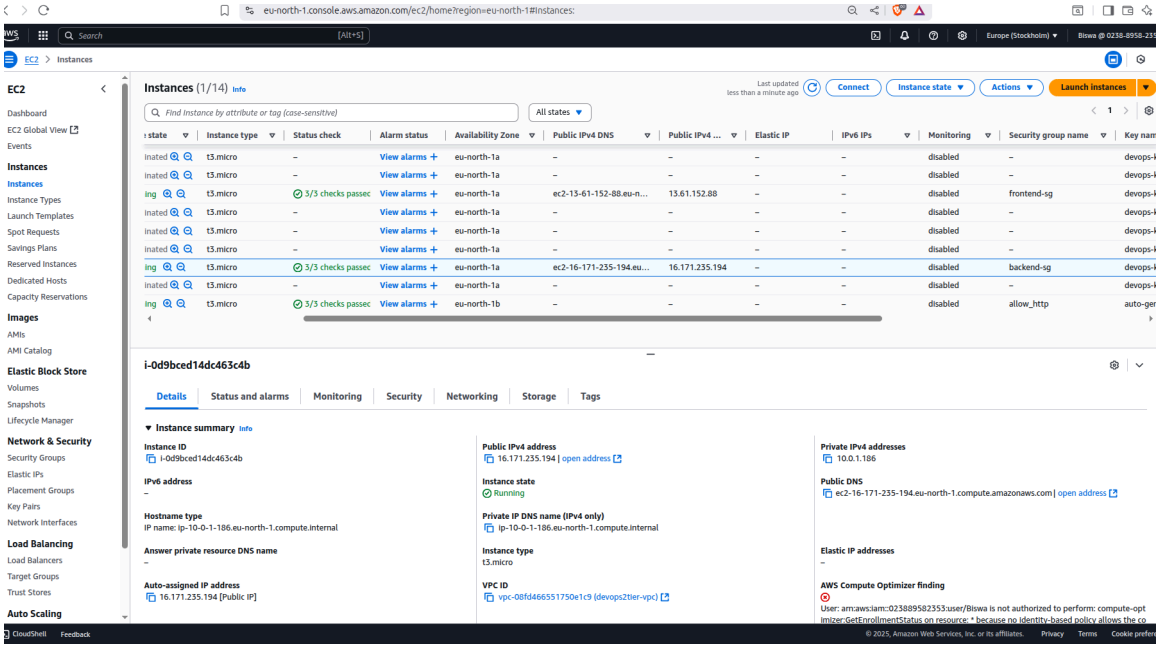
```
#!/bin/bash
sudo apt update -y
sudo apt install docker.io -y
sudo systemctl enable docker
sudo systemctl start docker
sudo usermod -aG docker ubuntu

Sudo docker run -d -p 3306:3306 biswaa18/backend-app:latest
```

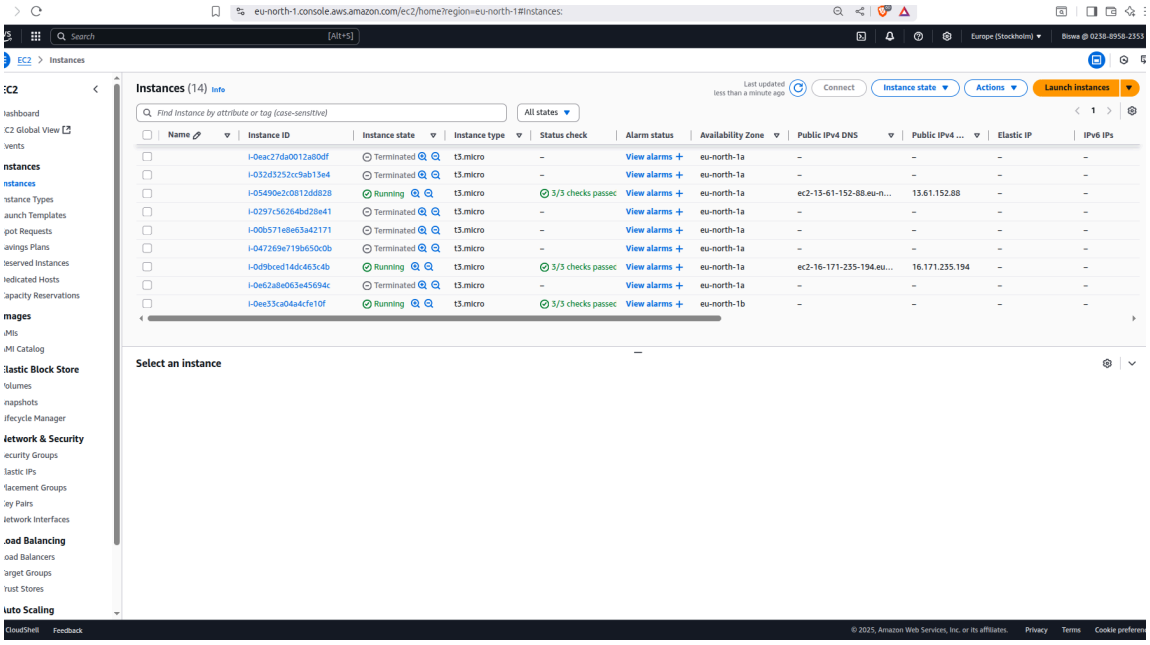
5. A Textfile containing terraform outputs.

```
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.7.0

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
vyper@vyper-MS-7C95:~/Desktop/2241018146_bisu/devops-2tier$
```

```
@vyper-MS-7C95:~/Desktop/2241018146_bisu/devops-2tier$ terraform apply -auto-approve
vpc.my_vpc: Refreshing state... [id=vpc-08fd466551750e1c9]
internet_gateway.igw: Refreshing state... [id=igw-021daa9e2efe0c9ee]
route_table.public_rt: Refreshing state... [id=rtb-02cb12ed3f0b831d0]
subnet.public_subnet: Refreshing state... [id=subnet-0a82047a45d199cb6]
security_group.backend_sg: Refreshing state... [id=sg-0ac092445a571cdb8]
security_group.frontend_sg: Refreshing state... [id=sg-034987e791f7e27a3]
route.public_internet_access: Refreshing state... [id=r-rtb-02cb12ed3f0b831d01080289494]
route_table_association.public_assoc: Refreshing state... [id=rtbassoc-00f437312cfaf1959]
instance.backend: Refreshing state... [id=i-047269e719b650c0b]
instance.frontend: Refreshing state... [id=i-032d3252cc9ab13e4]

form used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
update in-place
destroy and then create replacement

form will perform the following actions:

aws_instance.backend will be updated in-place
resource "aws_instance" "backend" {
    id                                   = "i-047269e719b650c0b"
  ~ public_dns                           = "ec2-16-170-163-36.eu-north-1.compute.amazonaws.com" -> (known after apply)
  ~ public_ip                            = "16.170.163.36" -> (known after apply)
    tags                                 = {}
  ~ user_data                            = <<-EOT
        #!/bin/bash
        apt update -y
        apt install -y docker.io
        systemctl start docker

        mkdir -p /home/ubuntu/backend
        cd /home/ubuntu/backend

        # Create Flask backend app
        cat <<EOT > app.py
        from flask import Flask, request, jsonify
        from flask_cors import CORS
        from pymongo import MongoClient
      + import sys

        app = Flask(__name__)
```

```
      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/sda1" -> null
          - encrypted             = false -> null
          - iops                  = 100 -> null
          - tags                  = {} -> null
          - tags_all              = {} -> null
          - throughput            = 0 -> null
          - volume_id             = "vol-0c0b025519c9f632c" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp2" -> null
            # (1 unchanged attribute hidden)
        }
    }

Plan: 1 to add, 1 to change, 1 to destroy.

Changes to Outputs:
  ~ frontend_public_ip = "51.20.78.46" -> (known after apply)
  ~ frontend_url       = "http://51.20.78.46:80" -> (known after apply)
aws_instance.frontend: Destroying... [id=i-032d3252cc9ab13e4]
aws_instance.frontend: Still destroying... [id=i-032d3252cc9ab13e4, 00m10s elapsed]
aws_instance.frontend: Still destroying... [id=i-032d3252cc9ab13e4, 00m20s elapsed]
aws_instance.frontend: Still destroying... [id=i-032d3252cc9ab13e4, 00m30s elapsed]
aws_instance.frontend: Destruction complete after 32s
aws_instance.backend: Modifying... [id=i-047269e719b650c0b]
aws_instance.backend: Still modifying... [id=i-047269e719b650c0b, 00m10s elapsed]
aws_instance.backend: Still modifying... [id=i-047269e719b650c0b, 00m20s elapsed]
aws_instance.backend: Still modifying... [id=i-047269e719b650c0b, 00m30s elapsed]
aws_instance.backend: Modifications complete after 36s [id=i-047269e719b650c0b]
aws_instance.frontend: Creating...
aws_instance.frontend: Still creating... [00m10s elapsed]
aws_instance.frontend: Creation complete after 15s [id=i-0e62a8e063e45694c]

Apply complete! Resources: 1 added, 1 changed, 1 destroyed.

Outputs:

frontend_public_ip = "13.60.224.55"
frontend_url = "http://13.60.224.55:80"
```


eu-north-1.console.aws.amazon.com/ec2/home?region=eu-north-1#Instances:
Search
[Alt+S]
Europe (Stockholm)
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores
Auto Scaling
Instances (1/14) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
IPv6 IPs
Monitoring
Security group name
t3.micro
View alarms
eu-north-1a
eu-north-1b
3/3 checks passed
ec2-13-61-152-88.eu-n...
13.61.152.88
ec2-16-171-235-194.eu...
16.171.235.194
disabled
frontend-sg
backend-sg
allow_http
i-0d9bced14dc463c4b
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0d9bced14dc463c4b
IPv6 address
Hostname type
IP name: ip-10-0-1-186.eu-north-1.compute.internal
Answer private resource DNS name
Auto-assigned IP address
16.171.235.194 [Public IP]
Public IPv4 address
16.171.235.194 | open address
Instance state
Running
Private IP DNS name (IPv4 only)
ip-10-0-1-186.eu-north-1.compute.internal
Instance type
t3.micro
VPC ID
vpc-08fd466551750e1c9 (devops2tier-vpc)
Private IPv4 addresses
10.0.1.186
Public DNS
ec2-16-171-235-194.eu-north-1.compute.amazonaws.com | open address
Elastic IP addresses
AWS Compute Optimizer finding
User: arn:aws:iam::023809582353:user/Biswa is not authorized to perform: compute-opt
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms


eu-north-1.console.aws.amazon.com/ec2/home?region=eu-north-1#Instances:
Search
[Alt+S]
Europe (Stockholm)
EC2 > Instances
Instances (14) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
IPv6 IPs
i-0eac27da0012a80df
i-032d3252cc9ab13e4
i-05490e2c0812dd828
i-0297c56264bd28e41
i-00b571e8e63a42171
i-047269e719b650c0b
i-0d9bced14dc463c4b
i-0e62a8e063e45694c
i-0ee33ca04a4cfe10f
Terminated
Running
t3.micro
3/3 checks passed
View alarms
eu-north-1a
eu-north-1b
ec2-13-61-152-88.eu-n...
13.61.152.88
ec2-16-171-235-194.eu...
16.171.235.194
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms

Docker-frontend

Dockerfiles (frontend)

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
EXPOSE 80
```

Index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>DevOps Login</title>
   <style>
      body {
         background-color: #282c34;
         color: #61dafb;
         font-family: Arial, sans-serif;
         text-align: center;
         margin: 0;
         padding: 0;
      }
      h1 {
         color: #61dafb;
         margin-top: 30px;
      }
      form {
         margin-top: 30px;
         background: #333;
         padding: 25px;
         border-radius: 8px;
         display: inline-block;
         text-align: left;
         width: 300px;
      }
      .form-header {
         font-size: 22px;
         font-weight: bold;
         text-align: center;
         margin-bottom: 15px;
      }
```

```
        label {
            display: block;
            margin-bottom: 5px;
        }
        input[type="email"], input[type="password"] {
            width: 100%;
            padding: 8px;
            border: 1px solid #61dafb;
            border-radius: 4px;
            margin-bottom: 15px;
            background: #222;
            color: #61dafb;
        }
        input[type="submit"] {
            width: 100%;
            background: #61dafb;
            border: none;
            padding: 10px;
            cursor: pointer;
            color: #000;
            font-weight: bold;
            margin-top: 5px;
        }
        .forgot {
            text-align: center;
            margin-top: 10px;
            font-size: 12px;
            color: #61dafb;
            cursor: pointer;
        }
        .success {
            margin-top: 15px;
            text-align: center;
            color: limegreen;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Welcome here ! Biswa this side</h1>
    <form id="loginForm">
        <div class="form-header">Login</div>

        <label for="email">Email:</label>
```

```
        <input type="email" id="email" name="email" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>

        <input type="submit" value="Submit">
        <div class="forgot">Forgot Password?</div>
        <div id="result" class="success"></div>
    </form>

    <script>
document.getElementById("loginForm").addEventListener("submit", async function(e) {
    e.preventDefault();
    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value;

    const response = await fetch("http://13.61.181.216:5000/save", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password })
    });

    const result = await response.json();
    document.getElementById("result").innerText = result.message;
});
</script>
</body>
</html>
```

docker-backend

- app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from pymongo import MongoClient

app = Flask(__name__)
CORS(app)

# MongoDB Atlas Connection
try:
```

```
    client =
MongoClient("mongodb+srv://dbuser:dbpass123@cluster0.i3gcqrs.mongodb.net/devopsdb?retr
yWrites=true&w=majority&appName=Cluster0")
    db = client["devopsdb"]
    users = db["users"]
    print("Connected to MongoDB Atlas successfully")
except Exception as e:
    print("MongoDB Connection Failed:", e)

@app.route('/save', methods=['POST'])
def save():
    data = request.json
    email = data.get("email")
    password = data.get("password")

    if not email or not password:
        return jsonify({"message": "Email and Password required"}), 400

    users.insert_one({"email": email, "password": password})
    return jsonify({"message": "Login data stored successfully!"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

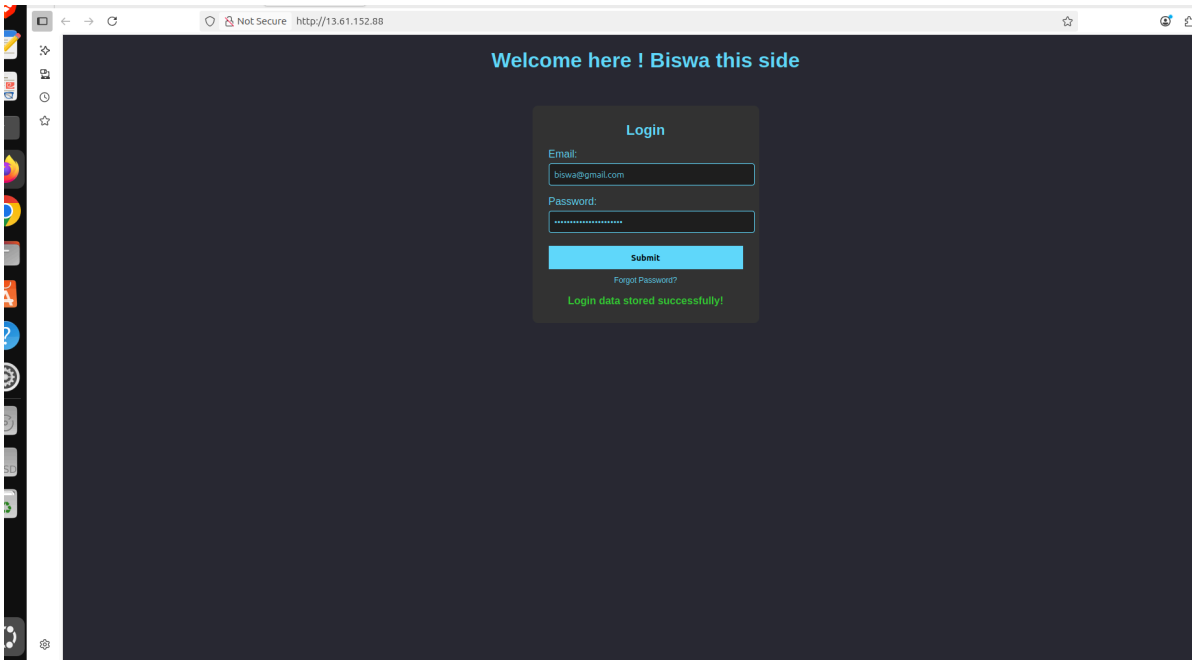
Requirements.txt

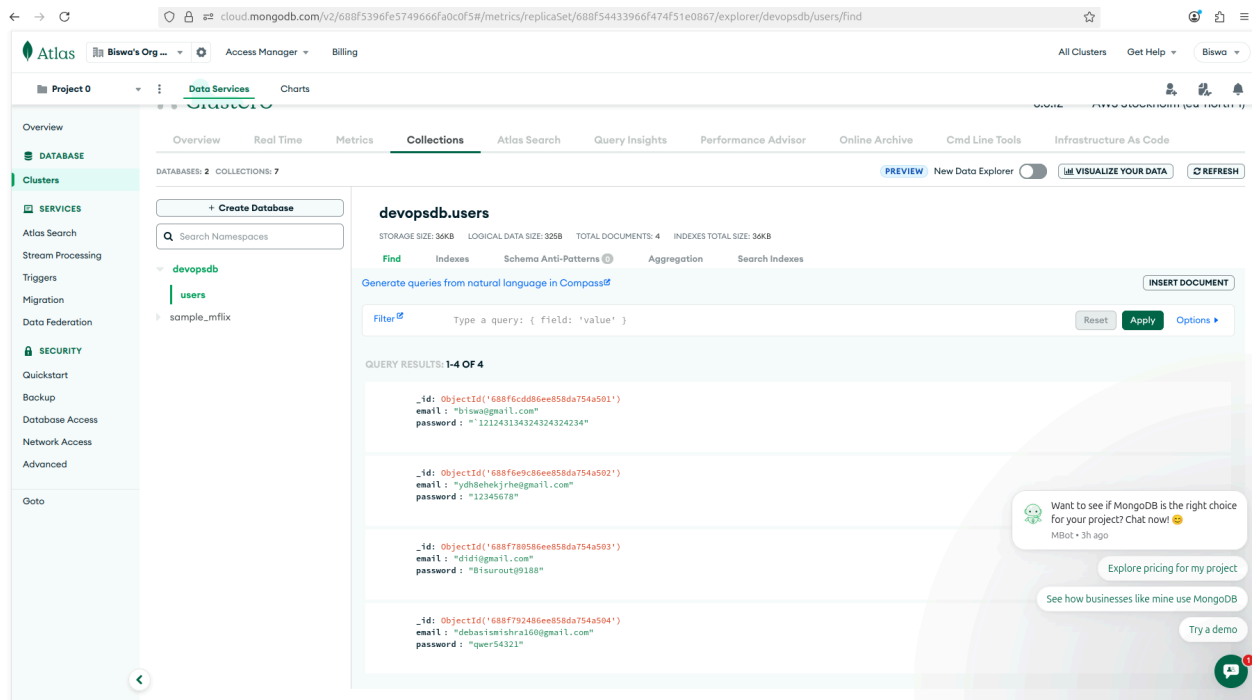
```
flask
Pymongo
```

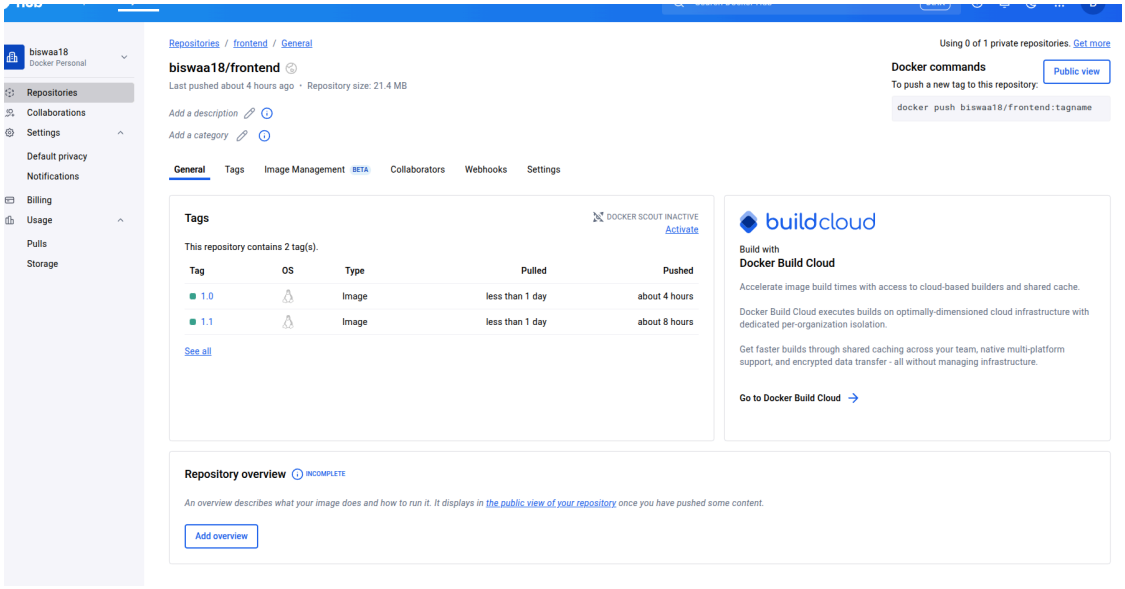
Dockerfiles (backend)

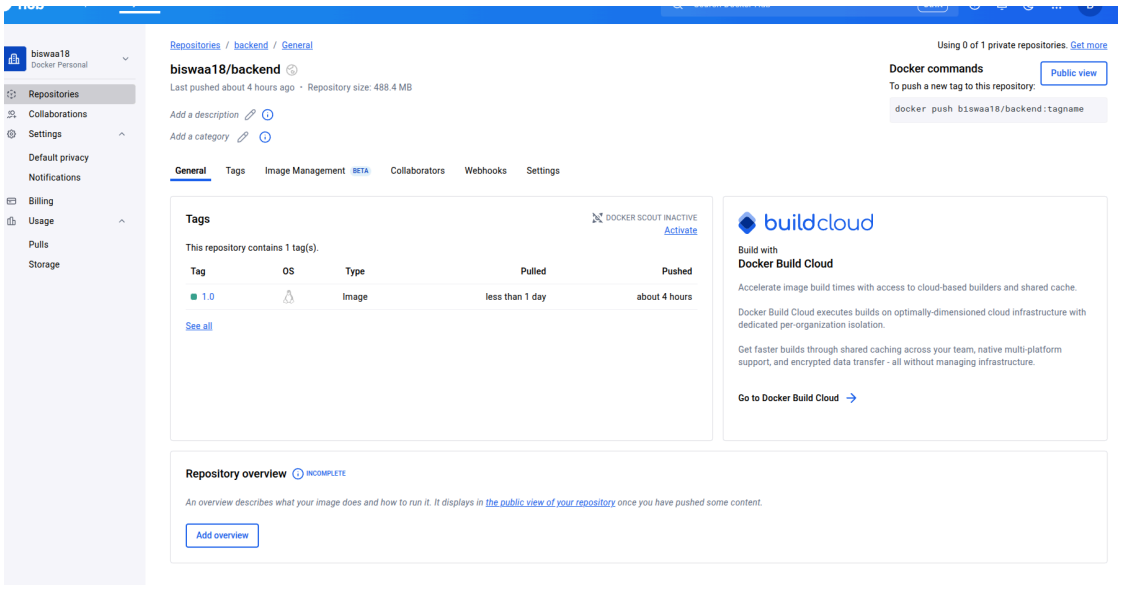
```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
RUN pip install flask pymongo flask-cors
COPY . .
CMD ["python", "app.py"]
```

6.Screenshots of Manual Testing






biswaa18
Docker Personal
Repositories
Collaborations
Settings
Default privacy
Notifications
Billing
Usage
Pulls
Storage
Repositories / frontend / General
biswaa18/frontend
Last pushed about 4 hours ago • Repository size: 21.4 MB
Add a description
Add a category
Using 0 of 1 private repositories. Get more
Docker commands
To push a new tag to this repository:
Public view
docker push biswaa18/frontend:tagname
General
Tags
Image Management BETA
Collaborators
Webhooks
Settings
Tags
DOCKER SCOUT INACTIVE
Activate
This repository contains 2 tag(s).
Tag
OS
Type
Pulled
Pushed
1.0
Image
less than 1 day
about 4 hours
1.1
Image
less than 1 day
about 8 hours
See all
buildcloud
Build with
Docker Build Cloud
Accelerate image build times with access to cloud-based builders and shared cache.
Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.
Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.
Go to Docker Build Cloud
Repository overview
INCOMPLETE
An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.
Add overview


biswaa18
Docker Personal
Repositories
Collaborations
Settings
Default privacy
Notifications
Billing
Usage
Pulls
Storage
Repositories / backend / General
biswaa18/backend
Last pushed about 4 hours ago • Repository size: 488.4 MB
Add a description
Add a category
Using 0 of 1 private repositories. Get more
Docker commands
To push a new tag to this repository:
Public view
docker push biswaa18/backend:tagname
General
Tags
Image Management BETA
Collaborators
Webhooks
Settings
Tags
DOCKER SCOUT INACTIVE
Activate
This repository contains 1 tag(s).
Tag
OS
Type
Pulled
Pushed
1.0
Image
less than 1 day
about 4 hours
See all
buildcloud
Build with
Docker Build Cloud
Accelerate image build times with access to cloud-based builders and shared cache.
Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.
Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.
Go to Docker Build Cloud
Repository overview
INCOMPLETE
An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.
Add overview