

# Set Up a Flask Web Application using s3 Bucket with SNS topic and create Email & SMS subscription

## Step 1: Set Up AWS Account

### Create an AWS Account:

Go to the AWS Console.

Create an AWS account and log in.

### Create an S3 Bucket:

Navigate to the S3 service.

Create a new S3 bucket, and note down the bucket name.

### Set Up AWS Access:

Create an IAM user with S3 access.

Note down the access key and secret key.

## Step 2: Set Up a Flask Web Application

### Install Flask:

```
sudo apt update -y
```

```
sudo apt upgrade -y
```

```
sudo apt install pip
```

```
pip install flask boto3
```

### Create a Flask App:

```
sudo vim app.py
```

```
# app.py
from flask import Flask, request, render_template
import boto3

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')
```

```

@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']

    s3 = boto3.client('s3', aws_access_key_id='YOUR_ACCESS_KEY',
aws_secret_access_key='YOUR_SECRET_KEY')

    s3.upload_fileobj(file, 'your-s3-bucket-name', file.filename)

    return 'File uploaded successfully!'

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')

```

### Create HTML Template:

**mkdir templates**

**cd templates**

**sudo vim index.html**

```

<!-- templates/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload to S3</title>
</head>
<body>
    <h1>Upload File to S3</h1>
    <form action="/upload" method="post" enctype="multipart/form-data">
        <input type="file" name="file" required>
        <button type="submit">Upload</button>

```

```
</form>

</body>

</html>
```

### **Step 3: Run the Flask App**

#### **Run the Flask App:**

**Python3 app.py**

Open your browser and go to <http://127.0.0.1:5000/> to access the web application.

#### **Upload a File:**

Choose a file in the web form and click "Upload."

#### **Important Note:**

This example uses a hardcoded access key and secret key for simplicity. In production, you should use environment variables or a more secure method to manage credentials.

Ensure your AWS credentials are secured and follow best practices.

The Flask development server is not suitable for production. Deploy the application using a production-ready server like Gunicorn or uWSGI.

This example lacks error handling, logging, and security measures. In a production environment, you should enhance the code accordingly.

# Dockerization Above Flask Application with s3

# For Docker Installation

```
sudo apt-get update  
sudo apt-get install docker.io -y  
sudo usermod -aG docker $USER && newgrp docker
```

# For Minikube & Kubectl

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube  
sudo snap install kubectl --classic  
minikube start --driver=docker
```

Dockerfile:

**sudo vim Dockerfile**

# Use an official Python runtime as a parent image

FROM python:3.8-slim

# Set the working directory to /app

WORKDIR /app

```
# Copy the current directory contents into the container at /app
COPY . /app
```

```
# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Make port 5000 available to the world outside this container
EXPOSE 5000
```

```
# Define environment variable
ENV FLASK_APP=app.py
```

```
# Run app.py when the container launches
CMD ["flask", "run", "--host=0.0.0.0"]
```

## **Building and Running the Docker Container:**

Create a requirements.txt File:

In your project directory, create a file named requirements.txt with the following content:

**Vim requirements.txt**

```
Flask
```

```
boto3
```

## **Build the Docker Image:**

Open a terminal, navigate to the directory containing your Dockerfile, and run the following command:

**`docker build -t flask-s3-app .`**

This command builds a Docker image named flask-s3-app.

### **Run the Docker Container:**

After successfully building the image, run the following command to start the Docker container:

**`docker run -p 5000:5000 flask-s3-app`**

This maps port 5000 on your host machine to port 5000 in the Docker container.

Access the Application:

Open your web browser and go to <http://localhost:5000>. You should be able to access your Flask application running inside the Docker container.

## **Kubernetes Above Flask Application with s3**

Certainly! Below is an example of a Kubernetes Deployment and Service configuration for your Flask application. This assumes you already have a running Kubernetes cluster. Save the following YAML configurations into separate files:

```
kubectl create secret docker-registry my-docker-credentials \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username= \
  --docker-password=
```

**`vim flask-s3-app-deployment.yaml:`**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-s3-app-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-s3-app
```

```
template:
  metadata:
    labels:
      app: flask-s3-app
  spec:
    containers:
      - name: flask-s3-app-container
        image: akashbkochure/flask-s3-app:latest
        ports:
          - containerPort: 5000
        imagePullSecrets:
          - name: my-docker-credentials
```

**vim flask-s3-app-service.yaml:**

```
apiVersion: v1
kind: Service
metadata:
  name: flask-s3-app-service
spec:
  selector:
    app: flask-s3-app
  ports:
    - protocol: TCP
      port: 5000
      nodePort: 31200
      targetPort: 5000
  type: NodePort
```



### **Deploy to Kubernetes:**

#### **Apply the Deployment:**

```
kubectl apply -f flask-s3-app-deployment.yaml
```

#### **Apply the Service:**

```
kubectl apply -f flask-s3-app-service.yaml
```

```
nohup kubectl port-forward svc/flask-s3-app-service 5000:5000 --address 0.0.0.0 > port-forward.log 2>&1 &
```

#### **Check the status of the deployment:**

```
kubectl get pods
```

Wait until the pod is in the "Running" state.

Once the pod is running, get the external IP of the service:

```
kubectl get service flask-s3-app-service
```

Note the external IP (it might take a moment for the external IP to be assigned).

Open your web browser and navigate to [http://EXTERNAL\\_IP](http://EXTERNAL_IP). You should be able to access your Flask application deployed on Kubernetes.

Please replace EXTERNAL\_IP with the actual external IP of your service.

This is a basic setup, and for production, you might consider using an Ingress controller for better routing and management of external access. Also, secure your application by using HTTPS and consider additional security measures depending on your deployment environment.

# Setup SNS With S3 Bucket

## 1. Create an SNS Topic

Navigate to the Simple Notification Service portion of the AWS Console and create an SNS Topic:

1. Go to the *SNS* dashboard in the AWS Console.
2. Create an *SNS Topic*.
3. (Eventually) edit the topic's resource policy so that the S3 bucket will be able to send events to that topic.

Ignore any KMS errors - the error won't impact the Hands-On lab.

## 2. Challenge

### Configure the Bucket — Part 1

Setting up events allows us to receive notifications whenever there are events happening in our bucket that we want to be notified about.

Navigate back in the S3 Management Console:

1. Configure your bucket settings.
2. *Event name*: "S3ObjectCreated".

We'll then receive an error: "Unable to validate the following destination configurations. Permissions on the destination do not allow S3 to publish notifications from this bucket." The reason we got this is because we haven't yet configured our SNS topic policy to accept notification events from our S3 bucket. Let's do that now.

## 3. Challenge

### Modify the SNS Topic Policy

Navigate to Simple Notifications Services Console.

Add the following code:

```
{  
  
  "Version": "2008-10-17",
```

```
"Id": "example-ID",

"Statement": [

  {

    "Sid": "s3-event-notifier",

    "Effect": "Allow",

    "Principal": {

      "Service": "s3.amazonaws.com"

    },

    "Action": [

      "SNS:Publish"

    ],

    "Resource": "arn:aws:sns:us-east-1:375736852765:S3FileUploadTopic",

    "Condition": {

      "ArnLike": {

        "aws:SourceArn": "arn:aws:s3:*:*:akashbkochure123"

      }

    }

  ]

}
```

In the `SNS:Publish` section, where it says, `SNS_ARN_REPLACE_ME`, paste in the SNS topic you added to that text file or note a minute ago. Then, copy and paste your S3 bucket ARN where it says `S3_BUCKET_ARN_REPLACE_ME`.

Make sure to leave the comma after the last bracket in our new statement, as this allows us to have more than one statement for this SNS topic's policy.

## Create the Bucket - Part 2

Navigate back in the S3 Management Console.

1. Configure bucket settings again.

This time, we should see we now have one active notification.

### 4. Challenge

## Create the Email Subscription and Create the SMS Subscription

### Create the Email Subscription

1. Navigate to Simple Notification Service.
2. Configure email settings.
3. Accept confirmation AWS email from inbox.
4. If successful, subscription confirmation page will appear.

### Create the SMS Subscription

5. Navigate to SNS Topics.
6. Select *S3Events*.
7. Insert "HELLO" under *Subject-optional*.
8. Insert "This is a test message" under *Message body*.
9. Save changes.
10. If successful, Amazon S3 Notification email sent to your inbox.