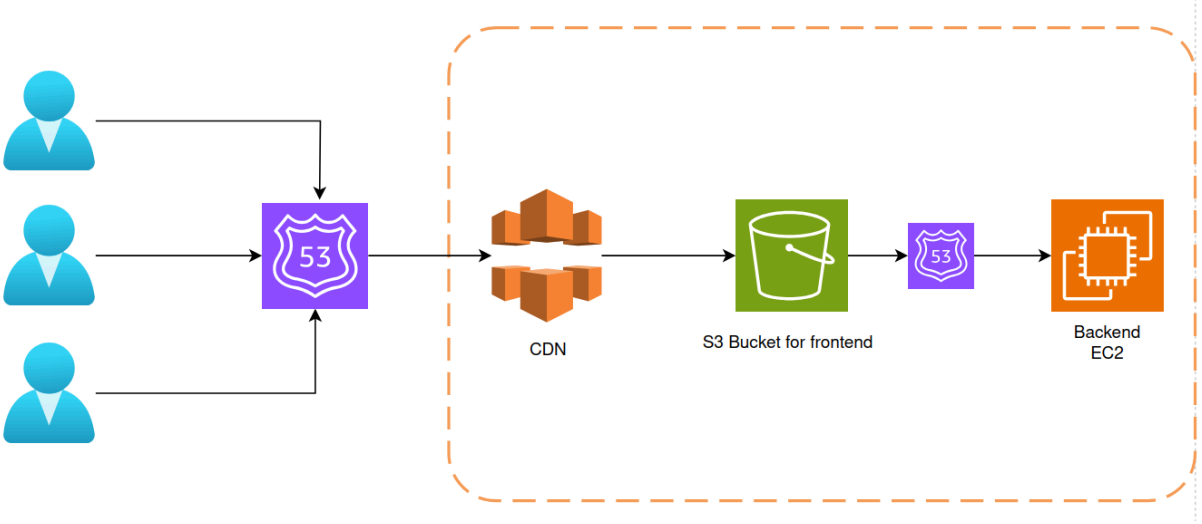
Deployment Automation

Pre-Requisites:

- AWS CLI installed (required for scripts to have access to AWS) **Guide**
- AWS CLI Access keys setup **Guide**
- The account used for CLI must have access to all the required resources

Simplified Diagram



Step 1:

We will create a ec2 instance with docker and nginx-proxy pre-installed for hosting our Backend APIs

To use this script:

- Make the script file executable with the following command

  ```
  chmod +x <path to script file>
  ```

- Provide the values as per your requirements in the first few lines of the script
- Run the script with the following command
  `./<script-file-name>`

Script for creating ec2 machine with docker and nginx proxy already installed

```
#!/bin/bash
```

```
# Define variables
REGION="us-east-1"                                  # Give your region
INSTANCE_TYPE="t2.micro"                            # Give instance type
SECURITY_GROUP="sg-0a49e97c7dfa3d678"               # Security group ARN
KEY_NAME="alkari-testing"                           # Pem key name (must be
already generated)
STORAGE_SIZE=30                                     # Storage in GB
AMI_ID="ami-04b4f1a9cf54c11d0"                      # Image ami ID
INSTANCE_NAME="alkari3"                             # Your instnace Name
# Dont change this
USER_DATA=$(cat <<EOF
#!/bin/bash
# Update system
apt update -y && apt upgrade -y
# Install Docker
apt install -y docker.io
systemctl start docker
systemctl enable docker
# Install Nginx Proxy
apt install -y nginx
systemctl start nginx
systemctl enable nginx
EOF
)
INSTANCE_ID=$(aws ec2 run-instances \
   --region "$REGION" \
   --image-id "$AMI_ID" \
   --instance-type "$INSTANCE_TYPE" \
   --key-name "$KEY_NAME" \
   --security-group-ids "$SECURITY_GROUP" \
   --block-device-mappings '[
       {"DeviceName": "/dev/xvda", "Ebs": {"VolumeSize":
'"$STORAGE_SIZE"', "VolumeType": "gp3"}}
   ]' \
   --user-data "$USER_DATA" \
   --query 'Instances[0].InstanceId' \
```

```
    --output text)
aws ec2 create-tags --resources "$INSTANCE_ID" --tags
Key=Name,Value="$INSTANCE_NAME"
echo "Instance created with ID: $INSTANCE_ID and Name: $INSTANCE_NAME"
```

Step 2:

- SSH into the EC2 machine we just created and setup your backend apis weather its through docker or is directly deployed
- Setup your nginx proxy as per your need
- Attach a ElasticIP to your backend machine if needed **Guide**
- You can also use a subdomain mapping for your backend APIs using route 53 A records
- Configure your frontend files to use your Backend APIs

Step 3:

For your frontend setup the S3 bucket with all your html and css files etc
Make sure the Folder Path you define in the script has the dir structure you want within the bucket and it should be the **absolute path**

To use this script:

- Make the script file executable with the following command

  `chmod +x <path to script file>`

- Replace the variables in the first couple lines of the script to your specific needs, make sure the BUCKET_NAME is unique globally
- Run the script with the following command
  `./<script-file-name>`

Script for setting up Static website hosting on your S3 for frontend

```
#!/bin/bash
# Variables
```

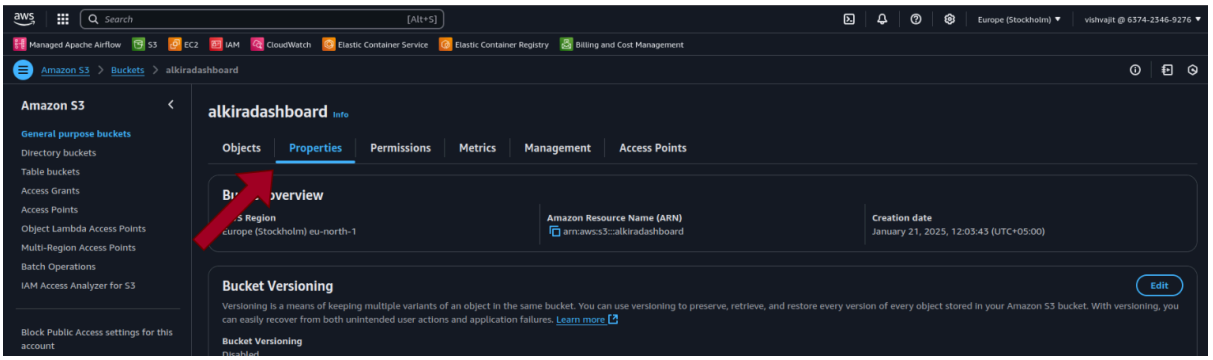
```
BUCKET_NAME="<your-unique-bucket-name>"
REGION="ap-south-1"
FOLDER_PATH="/alkari-automation/mywebapp"  # Change this to the folder
you want to upload
# Create S3 bucket
aws s3api create-bucket --bucket "$BUCKET_NAME" --region "$REGION"
--create-bucket-configuration LocationConstraint="$REGION"
# Enable public access block settings (disable block public access)
aws s3api delete-public-access-block --bucket "$BUCKET_NAME"
# Enable static website hosting
aws s3 website s3://$BUCKET_NAME/ --index-document index.html
--error-document error.html
# Create bucket policy for full public access
cat > bucket-policy.json <<EOL
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "PublicReadGetObject",
           "Effect": "Allow",
           "Principal": "*",
           "Action": "s3:GetObject",
           "Resource": "arn:aws:s3:::$BUCKET_NAME/*"
       }
   ]
}
EOL
# Apply the bucket policy
aws s3api put-bucket-policy --bucket "$BUCKET_NAME" --policy
file://bucket-policy.json
echo "S3 bucket '$BUCKET_NAME' created successfully with public access
and static website hosting enabled."
# Upload files recursively to the S3 bucket
aws s3 cp "$FOLDER_PATH" "s3://$BUCKET_NAME/" --recursive
# Clean up
rm bucket-policy.json
```

```
echo "S3 bucket '$BUCKET_NAME' created successfully with public access
and static website hosting enabled."
echo "Files from '$FOLDER_PATH' have been uploaded to
's3://$BUCKET_NAME/'"
```

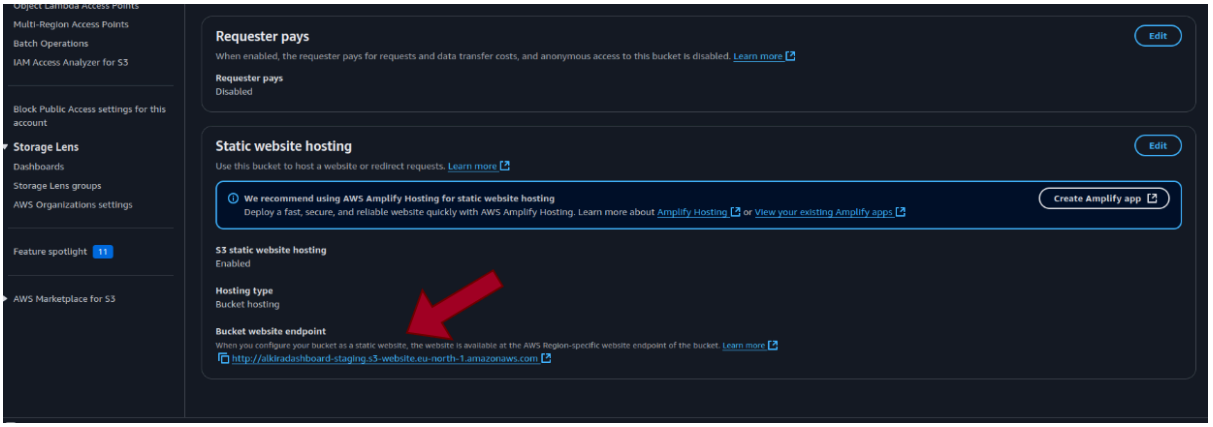
Step 4:

Now we will validate the functionality of our webapp

- In your AWS console go to the S3 service
- Click on your S3 bucket you just created (via script)
- Click on properties tab



- Scroll down till the end and there you will get your frontend url



- Check your applications functionality and interaction with the backend APIs

---

Step 5:

Now we will setup the CDN for our S3 bucket static website for that run the following script Give your values and run the script

Script for creating CDN with our S3 as its origin

```
#!/bin/bash
# Set variables
ORIGIN_NAME="<Your-S3-Bucket-Name>"  # Change this to your S3 bucket
name
REGION="ap-south-1"  # Change this to match your bucket's region
# Create CloudFront distribution
aws cloudfront create-distribution --distribution-config "{
   \"CallerReference\": \"$(date +%s)\",
   \"Origins\": {
       \"Quantity\": 1,
       \"Items\": [
           {
               \"Id\": \"$ORIGIN_NAME\",
               \"DomainName\":
\"$ORIGIN_NAME.s3-website.$REGION.amazonaws.com\",
               \"CustomOriginConfig\": {
                   \"HTTPPort\": 80,
                   \"HTTPSPort\": 443,
                   \"OriginProtocolPolicy\": \"http-only\"
               }
           }
       ]
   },
   \"DefaultCacheBehavior\": {
       \"TargetOriginId\": \"$ORIGIN_NAME\",
       \"ViewerProtocolPolicy\": \"redirect-to-https\",
       \"AllowedMethods\": {
           \"Quantity\": 2,
```

```
            \"Items\": [\"GET\", \"HEAD\"],
            \"CachedMethods\": {
                \"Quantity\": 2,
                \"Items\": [\"GET\", \"HEAD\"]
            }
        },
        \"ForwardedValues\": {
            \"QueryString\": false,
            \"Cookies\": {
                \"Forward\": \"none\"
            }
        },
        \"MinTTL\": 0,
        \"DefaultTTL\": 86400,
        \"MaxTTL\": 31536000
    },
    \"Comment\": \"S3 static website-backed CloudFront distribution\",
    \"Enabled\": true
}"
```

- Test your CDN by pasting your CDN domain name in a browser

Now we can setup alternate domain name and set custom behaviour, http / https settings or page refresh 404 issue

It depends as per use case

Step 6:

We can setup subdomain mapping for our CDN by creating a CNAME record type and give our CDN domain name without the http part