AWS Hands-on Detailed Guides

This document contains six step-by-step AWS implementation guides with exact console/CLI locations.

# 1. Hosting Static Websites on AWS S3 and EC2

1. **Create bucket**

* Name it exactly your domain if you plan to use one later (e.g., example.com).
* Choose region (e.g., ap-south-1 / ap-northeast-1).
* Uncheck “Block all public access” (we will make it public). Acknowledge warning.

1. **Enable static website hosting**

* In the bucket → **Properties** → **Static website hosting** → **Enable**.
* Index document: index.html
* Error document: error.html
* Note the **Website endpoint** URL (e.g., http://example.com.s3-website-ap-south-1.amazonaws.com).

1. **Upload site files**

* Put index.html, error.html, CSS/JS/assets into the bucket (top-level or in folders as you like).

1. **Make objects public**

* **Permissions** → **Bucket policy** → paste a public-read policy like:

{

"Version": "2012-10-17",

"Statement": [{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::example.com/\*"

}]

}

* Save. Now visiting the Website endpoint should load your site.

1. **(Optional) Map your domain (no HTTPS)**

* In Route 53 (or your DNS), create a record for example.com:
  + Type: **A**
  + **Alias** → target: **S3 website endpoint** (must pick the website endpoint, not the REST endpoint).
  + For www.example.com, add a CNAME to the same S3 website endpoint.
* This gives you http://example.com. **HTTPS requires CloudFront** (next section).

## A2. Production-grade S3 + CloudFront (HTTPS + private bucket)

Recommended for real sites. Bucket stays private, CloudFront serves content with free TLS.

1. **Create (or keep) your S3 bucket** (keep “Block all public access” **ON** this time—private bucket).
2. **Upload site files** to the bucket (same as before).
3. **Create a CloudFront distribution**

* **Origin domain**: choose your S3 bucket (select the **bucket’s REST endpoint**, not the website endpoint).
* **Origin access**: choose **Origin access control (OAC)** (newer and recommended).
* After creation, CloudFront will ask you to **update the bucket policy**. Use their suggested policy, or a model like this:

{

"Version": "2012-10-17",

"Statement": [{

"Sid": "AllowCloudFrontServicePrincipalReadOnly",

"Effect": "Allow",

"Principal": { "Service": "cloudfront.amazonaws.com" },

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::example.com/\*",

"Condition": {

"StringEquals": {

"AWS:SourceArn": "arn:aws:cloudfront::YOUR\_AWS\_ACCOUNT\_ID:distribution/DISTRIBUTION\_ID"

}

}

}]

}

* In CloudFront **Behaviors**, set:
  + **Viewer protocol policy**: Redirect HTTP to HTTPS.
  + **Compress objects automatically**: On.
* **Default root object**: index.html.
* **Custom domain + HTTPS**:
  + Under **Settings → Alternate domain names (CNAMEs)**: add example.com, www.example.com.
  + Under **SSL certificate**: choose **“Custom SSL certificate (ACM)”** and select a cert from **ACM in us-east-1** for your domains (create/validate via ACM if you don’t have one).

1. **DNS**

* In Route 53, create **A (Alias)** records pointing to your CloudFront distribution for example.com and www.example.com.

1. **Test**

* Open https://example.com — should be fast + HTTPS.

1. **Cache control (optional)**

* Add proper Cache-Control headers on objects (e.g., long TTL for assets, short/no cache for index.html if you deploy SPA).

# 2. EC2 Setup and MySQL Database Management

1. Launching an EC2 Instance:

* **Log in to AWS Management Console:** Access your AWS account.
* **Navigate to EC2:** Go to the EC2 dashboard.
* **Launch Instance:** Click "Launch Instance" and choose an appropriate Amazon Machine Image (AMI), such as Amazon Linux 2 or Ubuntu Server.
* **Choose Instance Type:** Select an instance type based on your performance and resource requirements.
* **Configure Instance Details:** Configure network, storage, and other settings as needed.
* **Add Storage:** Define the size and type of storage for your instance.
* **Configure Security Group:** Create or select a security group that allows inbound traffic on necessary ports (e.g., SSH for access, and later, port 3306 for MySQL).
* **Review and Launch:** Review your configuration and launch the instance, creating or selecting a key pair for secure SSH access.

2. Connecting to the EC2 Instance:

* **Connect via SSH:** Use an SSH client and your key pair to connect to the EC2 instance's public IP or DNS.

Code

ssh -i /path/to/your-key.pem ec2-user@your-ec2-public-ip

(Replace ec2-user with the appropriate username for your chosen AMI, e.g., ubuntu for Ubuntu AMIs.)

3. Installing MySQL Server:

Update System Packages.

Code

sudo yum update -y *# For Amazon Linux*  
 sudo apt update && sudo apt upgrade -y # For Ubuntu

Install MySQL Server.

Code

sudo yum install -y mysql-server *# For Amazon Linux*  
 sudo apt install -y mysql-server # For Ubuntu

Start and Enable MySQL Service.

Code

sudo systemctl start mysqld *# For Amazon Linux*  
 sudo systemctl enable mysqld *# For Amazon Linux*  
 sudo systemctl start mysql *# For Ubuntu*  
 sudo systemctl enable mysql # For Ubuntu

4. Securing MySQL Installation:

* Run mysql\_secure\_installation: This script guides you through setting a root password, removing anonymous users, disallowing remote root login, and removing test databases.

Code

sudo mysql\_secure\_installation

5. Managing MySQL Database:

Log in to MySQL.

Code

mysql -u root -p

Enter the root password you set during the secure installation. Create Databases and Users.

Code

CREATE DATABASE mydatabase;  
 CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';  
 GRANT ALL PRIVILEGES ON mydatabase.\* TO 'myuser'@'localhost';  
 FLUSH PRIVILEGES;

# 3. Web Application Deployment using AWS Elastic Beanstalk

1. Access Elastic Beanstalk. Log in to the AWS Management Console and search for and select "Elastic Beanstalk".
2. Create an application. Click the "Create Application" button.
3. Configure the environment:
   * Environment tier: Select Web server environment for web applications.
   * Application name: Enter a unique name for your application, such as MyWebApp.
   * Platform: Choose the platform that corresponds to your application's programming language and framework (e.g., Python, Node.js, Java).
   * Application code: Select the "Upload your code" option.
   * Upload your code: Choose your local .zip file or specify an S3 URL.
   * Version label: Provide a unique label for this version.
4. Review and launch. On the final review page, confirm your settings and click Submit or Create Application to start the deployment.
5. Monitor deployment. Elastic Beanstalk will begin provisioning the necessary resources. You can track the progress on the environment's dashboard.
6. Access your application. Once the environment's health status turns green, you can access your live web application by clicking the URL at the top of the dashboard.

# 4. Serverless Computing – S3 and Lambda Integration

# 1. Create an S3 Bucket

1. In AWS Console top search bar → type **S3** → click **S3** service.
2. On the **Buckets** page, click **Create bucket** (top right).
3. Fill in:
   * **Bucket name**: must be globally unique (e.g., my-file-processing-bucket).
   * **AWS Region**: pick the one closest to your users (e.g., Asia Pacific (Mumbai) ap-south-1).
4. Leave **Block Public Access** checked (recommended, unless you want public files).
5. Leave other defaults, click **Create bucket**.
6. You’ll see the bucket in the list; click its name to enter.

# 2. Create a Lambda Function

1. In AWS Console top search bar → type **Lambda** → click **Lambda** service.
2. On the Lambda dashboard, click **Create function**.
3. Choose **Author from scratch**.
   * **Function name**: e.g., S3FileProcessor.
   * **Runtime**: choose Python 3.11 (or Node.js 18.x if you prefer).
4. Under **Permissions**, leave “Create a new role with basic Lambda permissions”.
5. Click **Create function** (bottom).
6. On the function details page:
   * Go to the **Code** tab → inline editor.
   * Example Python code for testing:
   * import json
   * def lambda\_handler(event, context):
   * print("Event:", json.dumps(event))
   * return {"statusCode": 200, "body": "File processed!"}
   * Click **Deploy** (top right).

# 3. Configure the Lambda Function’s Execution Role

When you created the Lambda, AWS automatically created an **IAM role** (called something like S3FileProcessor-role-xxxx). This role must have **S3 permissions**.

1. In Lambda → **Configuration** tab → **Permissions** → click the role name (it opens IAM).
2. In IAM → Role summary → **Add permissions** → **Attach policies**.
3. Search for and attach:
   * **AmazonS3ReadOnlyAccess** (if Lambda only needs to read).
   * Or **AmazonS3FullAccess** (if it needs to read & write).
   * (Later you can create a custom least-privilege policy for only your bucket).

# 4. Set Up an S3 Trigger

1. Go back to your **S3 bucket** → click its name.
2. Inside the bucket → click the **Properties** tab (top navigation bar).
3. Scroll down to **Event notifications** → click **Create event notification**.
4. Fill in:
   * **Event name**: e.g., InvokeLambdaOnUpload.
   * **Prefix/Suffix** (optional):
     + Prefix = incoming/ (if you want only files in that folder).
     + Suffix = .jpg (if you want only images).
   * **Event types**: check **All object create events**.
5. In **Destination** → choose **Lambda function**.
   * Select the function you created (S3FileProcessor).
   * If asked, AWS will automatically add permission for S3 to invoke the Lambda.
6. Click **Save changes**.

# 5. Test the Integration

1. Go to your bucket → **Objects** tab → click **Upload**.
2. Add a test file (e.g., test.txt or image.jpg).
3. Click **Upload** (bottom).
4. The upload will generate an **S3 event**, which triggers your Lambda.
5. To confirm:
   * Go to **Lambda** → open your function → **Monitor** tab → **View logs in CloudWatch**.
   * Open the latest log stream → you should see the event JSON (contains bucket name, object key, size, etc.).
6. Once confirmed, replace the Lambda code with your actual logic (image resize, file conversion, metadata logging, etc.).

# 5. EC2 Auto Scaling using Launch Templates and Scaling Policies

# 1. Create a Launch Template

1. In the AWS Console search bar → type **EC2** → click **EC2** service.
2. In the left navigation pane → under **Instances** → click **Launch Templates**.
3. Click **Create launch template**.
4. Fill the form:
   * **Launch template name**: e.g., web-app-template.
   * **Template version description**: e.g., Initial version with Nginx.

### Configure instance details:

* **Amazon Machine Image (AMI):**
  + Click **Application and OS Images (Amazon Machine Image)** → choose Amazon Linux 2023 (or a custom AMI if you have one).
* **Instance type:**
  + e.g., t3.micro (free tier) or larger for production.
* **Key pair (login):**
  + If you want SSH access, select an existing key pair or create a new one.
* **Network settings:**
  + Select your **VPC**.
  + Choose at least one **subnet**.
  + Security groups:
    - Create or select one that allows **port 22 (SSH)** from your IP and **port 80 (HTTP)** / **443 (HTTPS)** from the internet.
* **Storage (volumes):**
  + Root volume defaults to 8 GB gp3. Adjust if needed.
* **Advanced details (optional):**
  + **IAM instance profile:** Attach a role if your app needs S3/CloudWatch access.
  + **User data:** (for bootstrap scripts). Example: install Nginx.
  + #!/bin/bash
  + dnf update -y
  + dnf install -y nginx
  + systemctl enable nginx
  + echo "Hello from $(hostname)" > /usr/share/nginx/html/index.html
  + systemctl start nginx

1. Scroll to the bottom → click **Create launch template**.

# 2. Create an Auto Scaling Group (ASG)

1. Go back to the **EC2 dashboard**.
2. In the left menu → under **Auto Scaling** → click **Auto Scaling Groups**.
3. Click **Create Auto Scaling group**.
4. Fill the form:
   * **Auto Scaling group name:** e.g., web-app-asg.
   * **Launch template:** choose the one you just created.

### Network settings:

* **VPC:** Select your app’s VPC.
* **Subnets:** Choose **at least 2 subnets in different Availability Zones** for high availability.

### Load balancing (optional, recommended for web apps):

* Select **Attach to an existing load balancer** → choose an **Application Load Balancer (ALB)** or create one.
* Create a **target group** if prompted (type: instances, protocol: HTTP:80).

### Group size:

* **Desired capacity:** (how many instances to start with, e.g., 2).
* **Minimum capacity:** (1 to ensure at least one is always running).
* **Maximum capacity:** (e.g., 6 to handle scale-out).

### Health checks:

* Select **ELB health checks** if using a load balancer (better than EC2 status checks alone).
* Health check grace period: 120 seconds (so app has time to boot).

Click **Next**.

# 3. Configure Scaling Policies

On the **Configure scaling policies** page, you choose how Auto Scaling adjusts capacity.

### A) Target Tracking Scaling Policy (recommended)

1. Select **Add policy** → **Target tracking scaling policy**.
2. **Metric type:** choose ASGAverageCPUUtilization or ALBRequestCountPerTarget.
3. **Target value:** e.g., 50% CPU.
4. Auto Scaling will add/remove instances to keep average CPU ~50%.

### B) Step Scaling or Simple Scaling (advanced)

1. Create **CloudWatch alarms** (e.g., CPU > 70% for 5 mins → scale out, CPU < 30% for 5 mins → scale in).
2. Attach these alarms to step scaling policies.
3. Define **Scaling adjustment** (e.g., add 1 instance, remove 1 instance).

Click **Next**.

# 4. Review and Create

1. Review: Launch template, VPC/subnets, scaling policies, health checks.
2. If correct → click **Create Auto Scaling group**.
3. You’ll be redirected to the ASG dashboard where you can:
   * See scaling activity history.
   * Check instances being created.

# 6.S3 Bucket File Management and Public Access Configuration\

1. Create an S3 Bucket (if you haven't already):

* Navigate to the Amazon S3 console.
* Choose "Create bucket."
* Provide a unique bucket name and select an AWS Region.
* Configure any additional settings as needed (e.g., Block Public Access settings – typically, you'd keep these enabled initially and modify them later for specific public access needs).
* Choose "Create bucket."

2. Upload Files to the Bucket:

* Select the bucket you want to upload files to.
* Choose "Upload."
* Drag and drop or browse for the files you want to upload.
* Configure any upload options (e.g., storage class, encryption).
* Choose "Upload."

3. Configure Public Access for Specific Objects (Files):

* **Disable Bucket-Level Block Public Access (if necessary):**

If you intend to make some objects publicly accessible, you might need to adjust the bucket's "Block Public Access" settings. Navigate to the bucket's "Permissions" tab, then "Block public access (bucket settings)," and edit the settings to allow for object ACLs or public bucket policies if required for your specific public access strategy.

* **Grant Public Read Access to an Object:**
  + Select the object (file) you want to make public.
  + Choose the "Actions" dropdown and select "Make public" (if available and bucket settings allow).
  + Alternatively, navigate to the object's "Permissions" tab.
  + In the "Access control list (ACL)" section, choose "Edit."
  + For the "Everyone (public access)" grantee, check the "Read" permission for "Objects."
  + Acknowledge the warning about public access and choose "Save changes."

4. Configure Public Access using Bucket Policies (for more granular control):

* Navigate to the bucket's "Permissions" tab.
* Choose "Bucket policy."
* Write a JSON policy that grants public read access to specific objects or prefixes within the bucket. For example, to make all objects in a specific folder public:

Code

{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Effect": "Allow",  
 "Principal": "\*",  
 "Action": "s3:GetObject",  
 "Resource": "arn:aws:s3:::your-bucket-name/public-folder/\*"  
 }  
 ]  
 }

(Replace your-bucket-name and public-folder with your actual bucket name and desired folder.) Save the bucket policy.