← BrowserStack Integration

CI/CD & Test Reporting →

AWS for Test Automation

Duration: 5 Hours (Instruction) + 3 Hours (Hands-on)
Prerequisites: Basic understanding of test automation frameworks
Target Audience: QA Engineers with no cloud experience

Learning Objectives

By the end of this session, you will:

- Understand cloud computing basics from a QA perspective
- Set up AWS account with proper security
- Deploy test automation on EC2 instances
- Store and manage test artifacts in S3
- Use Lambda for scheduled smoke tests
- Integrate AWS with CI/CD pipelines
- Optimize costs and monitor usage

Cloud Computing Fundamentals for QA Engineers

```
 Your Office → Physical Servers → Install OS → Setup Software → Run Tests
Problems:
- Expensive upfront cost ($10,000+ for servers)
- Limited scalability (max 10 parallel tests)
- Maintenance overhead (updates, patches)
- Space and power costs
```

Cloud Computing:

```
 Laptop → Internet → Cloud Provider → Virtual Machines → Run Tests
Benefits:
- Pay only for what you use ($5-50/month)
- Scale instantly (1 to 1000+ parallel tests)
- No maintenance (provider manages hardware)
- Available worldwide (test from any region)
```

Why QA Engineers Need Cloud?

Problem 1: Slow Test Execution

```
 Traditional: 500 tests × 2 minutes = 1000 minutes (16+ hours)
Cloud: 500 tests ÷ 10 machines = 100 minutes (1.6 hours)
```

Problem 2: Limited Resources

```
 Office: 5 laptops = 5 parallel tests max
Cloud: 100 instances = 100 parallel tests
```

Problem 3: Cross-Browser Testing

```
 Office: Install Chrome, Firefox, Safari manually
Cloud: Pre-configured with all browsers
```

Problem 4: CI/CD Integration

Cloud Service Models

1. IaaS (Infrastructure as a Service)

- **What:** Virtual machines, storage, networking
- **Example:** AWS EC2, Azure VMs
- **QA Use:** Run Selenium Grid, Playwright tests
- **You Manage:** OS, software, tests
- **Provider Manages:** Hardware, networking

2. PaaS (Platform as a Service)

- **What:** Ready platform for apps
- **Example:** AWS Elastic Beanstalk, Heroku
- **QA Use:** Deploy test APIs, mock servers
- **You Manage:** Application, tests
- **Provider Manages:** OS, runtime, hardware

3. SaaS (Software as a Service)

- **What:** Ready-to-use software
- **Example:** BrowserStack, Sauce Labs
- **QA Use:** Cloud testing platforms
- **You Manage:** Test scripts only
- **Provider Manages:** Everything else

For QA Engineers, we primarily use IaaS (EC2) and SaaS (BrowserStack)

AWS vs Traditional Setup

Initial Cost	$10,000+	$0 (Free tier)
Monthly Cost	$500+	$20-100
Setup Time	1-2 weeks	15 minutes
Scalability	Limited to hardware	Unlimited
Maintenance	Your responsibility	AWS manages
Availability	Office hours only	24/7
Global Testing	One location	25+ regions

AWS Basics - Step by Step

Understanding AWS Services for Testing

Think of AWS as a supermarket for IT resources:

Compute (EC2) = Virtual Computers

- Like renting laptops in the cloud
- Use for: Running Playwright, Cypress, Selenium
- Pricing: ~$0.01/hour (t2.micro)

Storage (S3) = Cloud File Cabinet

- Like Google Drive for test artifacts
- Use for: Reports, screenshots, videos, logs
- Pricing: $0.023/GB/month

Functions (Lambda) = Pay-per-run Code

- Pricing: $0.20 per 1M requests

Monitoring (CloudWatch) = Dashboard

- Like Google Analytics for infrastructure
- Use for: Logs, metrics, alerts
- Pricing: $0.30/GB logs

AWS Global Infrastructure

AWS has **25+ regions** worldwide:

```
 US: us-east-1 (Virginia), us-west-2 (Oregon)
Europe: eu-west-1 (Ireland), eu-central-1 (Frankfurt)
Asia: ap-south-1 (Mumbai), ap-southeast-1 (Singapore)
```

Why this matters for QA:

- Test latency from different regions
- Compliance (data must stay in specific countries)
- Cost optimization (prices vary by region)

Best Practice: Start with **us-east-1** (cheapest, most services)

Setting Up AWS Account (For Beginners)

Step 1: Create AWS Account

Complete Guide:

1. **Visit** https://aws.amazon.com
2. **Click** "Create an AWS Account"

```
Email: your.email@company.com
Account Name: TestAutomation-YourName
Password: Strong password (save it!)
```

4. **Contact Information:**

```
✓ Select: Professional
Company: Your Company Name
Phone: +91-XXXXXXXXXX (with country code)
Address: Complete address
```

5. **Payment Method:**
   - Add credit/debit card
   - AWS charges $1 (refunded) to verify
   - **Don't worry:** Free tier = $0 charges for 12 months
6. **Identity Verification:**
   - Receive automated call/SMS
   - Enter 4-digit code shown on screen
7. **Select Support Plan:**
   - Choose: **Basic Support (Free)**
   - Don't select Developer ($29/month) yet
8. **Congratulations!** Account created

Important: You'll receive:

- Account ID (12 digits) - Save this!
- Root user email - Don't use for daily work!

Understanding Free Tier

12 Months Free (from signup date):

```
S3: 5 GB storage
    = ~5000 test reports
    = ~500 screenshots
    = ~50 test videos

Lambda: 1 Million requests/month
        = 100 health checks every 5 minutes for free
```

Always Free (forever):

```
 Lambda: 1M requests/month
CloudWatch: 10 custom metrics, 10 alarms
DynamoDB: 25 GB storage
```

Step 2: Secure Your Account (CRITICAL!)

Problem: Root account has unlimited access

- Can delete everything
- Can spend unlimited money
- High security risk

Solution: Create IAM users

Security Analogy:

```
 Root Account = Master Key to Office
IAM User = Individual Employee Badge
```

Enable MFA (Multi-Factor Authentication):

1. **Install Authenticator App:**
    - Google Authenticator (Mobile)
    - Authy (Mobile/Desktop)

```
AWS Console → Account (top right)
→ Security Credentials → MFA
→ Activate MFA → Virtual MFA device
→ Scan QR code with app
→ Enter two consecutive codes
```

3. **Now login requires:**
   - Password + 6-digit code from phone
   - Much more secure!

Step 3: Create IAM User for Daily Work

Why: Never use root account for testing!

Create Test Automation User:

1. **Navigate:** IAM Dashboard → Users → Add users
2. **User Details:**

```
Username: test-automation-user
Access type:
  ✓ Programmatic access (for CLI/scripts)
  ✓ AWS Management Console access (for web)

Console password:
  ○ Autogenerated password
  ✓ Custom password: YourStrongPassword123!

✓ Require password reset (uncheck for learning)
```

3. **Set Permissions:**

   For Learning (Full Access):

```
Attach existing policies directly:
✓ AdministratorAccess
```

```
Create policy with only needed permissions:
- EC2: Start/Stop instances
- S3: Read/Write specific bucket
- Lambda: Invoke functions
- CloudWatch: View logs
```

4. **Tags (Optional but Recommended):**

```
Key: Department, Value: QA
Key: Project, Value: TestAutomation
```

5. **Review and Create**

6. **IMPORTANT:** Download credentials.csv

```
User,Password,Access key ID,Secret access key,Console login link
test-automation-user,Pass123!,AKIAIOSFODNN7EXAMPLE,wJal...
```

Save this file securely! You can't retrieve it later.

Step 4: Install AWS CLI

AWS CLI = Command line tool to manage AWS from terminal

Check if Already Installed:

```
aws --version
# If installed: aws-cli/2.x.x ...
# If not: command not found
```

Installation:

Windows:

```
# Download installer from:
# https://awscli.amazonaws.com/AWSCLIV2.msi
```

```
# Verify:
aws --version
```

macOS:

```
 # Using Homebrew:
brew install awscli

# Or direct download:
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /

# Verify:
aws --version
```

Linux (Ubuntu/Debian):

```
 # Download installer
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscli

# Unzip
unzip awscliv2.zip

# Install
sudo ./aws/install

# Verify
aws --version
```

Step 5: Configure AWS CLI

Basic Configuration:

```
 aws configure

# You'll be prompted for:
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfi...
```

Where to find credentials:

- Access Key ID: From downloaded credentials.csv
- Secret Access Key: From downloaded credentials.csv
- Region: us-east-1 (cheapest, most services)
- Output: json (most common)

Verify Configuration:

```
 # Check who you are
aws sts get-caller-identity

# Output:
{
    "UserId": "AIDAI3QEXAMPLE",
    "Account": "123456789012",
    "Arn": "arn:aws:iam::123456789012:user/test-automation-user"
}

# List S3 buckets (should be empty for new account)
aws s3 ls

# Check EC2 instances (should be empty)
aws ec2 describe-instances
```

Configuration Files Location:

```
 ~/.aws/credentials  (Linux/Mac)
%USERPROFILE%\.aws\credentials  (Windows)

Contains:
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfi...
```

What is EC2?

EC2 = Elastic Compute Cloud = Virtual Computers in Cloud

Real-World Analogy:

```
 Buying a Laptop:
- Cost: $1000
- Fixed specs (8GB RAM, 256GB)
- Depreciates over time
- You maintain it

Renting EC2:
- Cost: $20/month
- Changeable specs (upgrade anytime)
- Stop paying when not using
- AWS maintains it
```

When to Use EC2 for Testing?

Scenario	Use EC2?	Why?
Daily regression (500+ tests)	✅ Yes	Cost-effective, scalable
CI/CD automated tests	✅ Yes	Always available
Load testing	✅ Yes	Scale to 100+ instances
Local quick testing	❌ No	Use local machine
BrowserStack already used	❌ No	Duplicate effort

EC2 Instance Types Explained

Instance Naming: `t2.micro`

- `micro` = Size (nano < micro < small < medium < large)

For Test Automation:

```
 t2.micro (Free Tier):
├── 1 vCPU, 1 GB RAM
├── Best for: Small test suites (50-100 tests)
├── Cost: FREE (750 hours/month)
└── Use case: Learning, small projects

t2.small:
├── 1 vCPU, 2 GB RAM
├── Best for: Medium suites (100-300 tests)
├── Cost: $0.023/hour = $17/month
└── Use case: Daily regression

t2.medium:
├── 2 vCPU, 4 GB RAM
├── Best for: Large suites (300-500 tests)
├── Cost: $0.046/hour = $34/month
└── Use case: Parallel execution (2-3 browsers)

t2.large:
├── 2 vCPU, 8 GB RAM
├── Best for: Heavy suites (500+ tests)
├── Cost: $0.094/hour = $69/month
└── Use case: Parallel execution (4-5 browsers)

c5.xlarge (Compute Optimized):
├── 4 vCPU, 8 GB RAM
├── Best for: Performance testing
├── Cost: $0.17/hour = $125/month
└── Use case: Maximum parallel execution
```

Cost Optimization Tips:

```
 Strategy 1: Stop when not testing
- Running 24/7: $69/month (t2.large)
- Running 8 hrs/day: $23/month (67% savings!)

Strategy 2: Use Spot Instances
- On-demand: $0.094/hour
```

```
Strategy 3: Reserved Instances (if used daily)
- 1-year commitment: 40% discount
- 3-year commitment: 60% discount
```

Launching Your First EC2 Instance

Method 1: AWS Console (Visual, Beginner-Friendly)

Step-by-Step:

1. **Navigate to EC2:**

```
AWS Console → Services → EC2 → Dashboard
```

2. **Launch Instance:**

```
Click: Launch Instance (Orange button)
```

3. **Name Your Instance:**

```
 Name: PlaywrightTestServer
Tags:
  Key: Environment, Value: Testing
  Key: Project, Value: Automation
```

4. **Choose AMI (Operating System):**

```
 ✓ Ubuntu Server 22.04 LTS (Free tier eligible)

Other options:
- Amazon Linux 2023 (AWS optimized)
- Windows Server (not free tier)
- Red Hat, CentOS (enterprise)
```

5. **Choose Instance Type:**

6. **Create Key Pair (IMPORTANT!):**

```
This is how you'll SSH into the instance!

Click: Create new key pair

Key pair name: playwright-test-key
Key pair type: RSA
Private key format: .pem (for Mac/Linux)
                    .ppk (for Windows/PuTTY)

Click: Create key pair

⚠️ SAVE THE FILE! You can't download it again!
Save to: ~/Downloads/playwright-test-key.pem
```

7. **Configure Network:**

```
Create new security group:

Security group name: test-automation-sg
Description: Security group for test automation

Inbound rules:
✓ SSH (port 22) from My IP (your current IP)
✓ HTTP (port 80) from Anywhere (optional - for viewing reports)
```

8. **Configure Storage:**

```
✓ 8 GB gp3 (Free tier eligible)

For heavy testing:
- 30 GB gp3 (for lots of screenshots/videos)
```

9. **Advanced Details (Optional):**

```
You can add a startup script to auto-install software
(We'll do this manually first for learning)
```

```
Click: Launch instance

Wait 2-3 minutes for instance to initialize

Status: Running ✓
```

Method 2: AWS CLI (Fast, Scriptable)

```
# Step 1: Create key pair and save to file
aws ec2 create-key-pair   --key-name playwright-test-key   --query 'KeyMat

# Set proper permissions (Mac/Linux only)
chmod 400 playwright-test-key.pem

# Step 2: Create security group
aws ec2 create-security-group   --group-name test-automation-sg   --descri

# Get your public IP
MY_IP=$(curl -s http://checkip.amazonaws.com)

# Step 3: Allow SSH from your IP
aws ec2 authorize-security-group-ingress   --group-name test-automation-sg

# Step 4: Launch instance
aws ec2 run-instances   --image-id ami-0c7217cdde317cfec \  # Ubuntu 22.04
  --count 1   --instance-type t2.micro   --key-name playwright-test-key
```

Connecting to Your EC2 Instance

Find Instance IP:

Console Method:

```
EC2 Dashboard → Instances → Click your instance
→ Copy "Public IPv4 address"
Example: 54.123.45.67
```

CLI Method:

```
# Output:
# ---------------------------
# |  DescribeInstances        |
# +----------------+----------+
# |  54.123.45.67  |  running  |
# +----------------+----------+
```

SSH Connection:

Mac/Linux:

```
 # Navigate to key file location
cd ~/Downloads

# Connect
ssh -i playwright-test-key.pem ubuntu@54.123.45.67

# First time you'll see:
# Are you sure you want to continue connecting (yes/no)?
# Type: yes

# You're now in the EC2 instance!
ubuntu@ip-172-31-xx-xx:~$
```

Windows (PowerShell):

```
 # Using OpenSSH (Windows 10+)
ssh -i playwright-test-key.pem ubuntu@54.123.45.67

# Or using PuTTY:
# Convert .pem to .ppk using PuTTYgen
# Open PuTTY
# Host: ubuntu@54.123.45.67
# Connection → SSH → Auth → Browse .ppk file
# Click Open
```

Troubleshooting Connection Issues:

```
 # Error: Permission denied (publickey)
# Solution: Check key file permissions
```

```
# Solution: Check security group allows SSH from your IP
aws ec2 describe-security-groups --group-names test-automation-sg

# Error: Connection refused
# Solution: Instance might still be initializing, wait 1-2 minutes
```

Setting Up Test Environment on EC2

Manual Setup (Learning)

Once connected via SSH:

```
 # Update system packages
sudo apt update && sudo apt upgrade -y

# Install Node.js 20.x
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Verify installation
node --version  # Should show v20.x.x
npm --version   # Should show 10.x.x

# Install Git
sudo apt install -y git
git --version

# Install Chrome
wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64
sudo apt install -y ./google-chrome-stable_current_amd64.deb
google-chrome --version

# Install Playwright
npm install -g @playwright/test
npx playwright install chromium firefox webkit
npx playwright install-deps

# Create workspace
mkdir -p ~/test-automation
```

```
npx playwright --version
```

Automated Setup (Production)

Create `setup-ec2.sh` :

```
#!/bin/bash
set -e  # Exit on any error

echo "🚀 Setting up Test Automation Environment..."

# Update system
echo "📦 Updating system packages..."
sudo apt update && sudo apt upgrade -y

# Install Node.js
echo "📦 Installing Node.js 20.x..."
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Install Git
echo "📦 Installing Git..."
sudo apt install -y git

# Install Chrome
echo "🌐 Installing Google Chrome..."
wget -q https://dl.google.com/linux/direct/google-chrome-stable_current_am
sudo apt install -y ./google-chrome-stable_current_amd64.deb
rm google-chrome-stable_current_amd64.deb

# Install Playwright globally
echo "🎭 Installing Playwright..."
sudo npm install -g @playwright/test

# Install browsers
echo "🌐 Installing Playwright browsers..."
npx playwright install chromium firefox webkit

# Install browser dependencies
echo "📦 Installing browser dependencies..."
npx playwright install-deps

# Create workspace
```

```
# Print versions
echo "✅ Setup complete!"
echo "📊 Installed versions:"
echo "Node: $(node --version)"
echo "npm: $(npm --version)"
echo "Chrome: $(google-chrome --version)"
echo "Playwright: $(npx playwright --version)"
echo ""
echo "💡 Navigate to workspace: cd ~/test-automation"
```

Copy and run on EC2:

```
 # On your local machine:
scp -i playwright-test-key.pem setup-ec2.sh ubuntu@54.123.45.67:~/

# On EC2:
chmod +x setup-ec2.sh
./setup-ec2.sh
```

Running Tests on EC2

Example 1: Simple Test

Create test on EC2:

```
 # SSH into EC2
ssh -i playwright-test-key.pem ubuntu@54.123.45.67

# Navigate to workspace
cd ~/test-automation

# Initialize Node project
npm init -y

# Install Playwright locally
npm install @playwright/test
```

```
const { test, expect } = require('@playwright/test');

test('google search', async ({ page }) => {
  await page.goto('https://www.google.com');
  await page.locator('[name="q"]').fill('Playwright');
  await page.locator('[name="q"]').press('Enter');
  await expect(page).toHaveTitle(/Playwright/);
});
EOF

# Run test
npx playwright test

# View report
npx playwright show-report
```

Example 2: Deploy Your Actual Tests

On your local machine:

```
 # Navigate to your test project
cd ~/my-playwright-tests

# Package your tests (excluding node_modules)
tar -czf tests.tar.gz   --exclude='node_modules'   --exclude='playwright-r

# Copy to EC2
scp -i playwright-test-key.pem tests.tar.gz ubuntu@54.123.45.67:~/test-aut

# SSH to EC2
ssh -i playwright-test-key.pem ubuntu@54.123.45.67

# Extract and setup
cd ~/test-automation
tar -xzf tests.tar.gz
npm install

# Run tests
npm test
```

Example 3: Scheduled Tests

```
 # Edit crontab
crontab -e

# Add line (daily at 9 AM UTC):
0 9 * * * cd ~/test-automation && npm test > ~/logs/test-$(date +\%Y\%m\%d

# Create logs directory
mkdir -p ~/logs

# View cron logs
ls -lh ~/logs/
cat ~/logs/test-20240130.log
```

S3 for Test Artifacts - Deep Dive

What is S3?

S3 = Simple Storage Service = Cloud Hard Drive

Key Concepts:

```
 Bucket = Folder (top level)
├── Contains: Objects (files)
├── Name: Globally unique (test-automation-yourname)
├── Region: us-east-1, ap-south-1, etc.
└── Access: Public or Private

Object = File
├── Key: reports/2024-01-30/index.html
├── Value: File content
├── Metadata: Content type, test-run, etc.
└── Size: 0 bytes to 5 TB
```

Creating S3 Bucket

Method 1: Console

```
   Name: test-automation-artifacts-yourname
   Region: us-east-1 (same as EC2)

3. Object Ownership:
   ✓ ACLs disabled (recommended)

4. Block Public Access:
   ✓ Block all public access (keep private)

5. Bucket Versioning:
   ✓ Enable (keeps file history)

6. Encryption:
   ✓ Server-side encryption (SSE-S3)

7. Create bucket
```

Method 2: CLI

```
 # Create bucket
aws s3 mb s3://test-automation-artifacts-yourname --region us-east-1

# Enable versioning
aws s3api put-bucket-versioning   --bucket test-automation-artifacts-yourn

# Enable encryption
aws s3api put-bucket-encryption   --bucket test-automation-artifacts-yourn
    "Rules": [{
      "ApplyServerSideEncryptionByDefault": {
        "SSEAlgorithm": "AES256"
      }
    }]
  }'
```

Organizing Test Results

Recommended Folder Structure:

```
 s3://test-automation-artifacts-yourname/
|
```

```
|   |   |   ├── run-001/
|   |   |   |   ├── index.html
|   |   |   |   └── assets/
|   |   |   └── run-002/
|   |   ├── 2024-01-31/
|   |   └── latest/  (symlink to most recent)
|   |
|   ├── cypress/
|   └── allure/
|
├── screenshots/
|   ├── 2024-01-30/
|   |   ├── test1-failed.png
|   |   └── test2-failed.png
|   └── 2024-01-31/
|
├── videos/
|   ├── 2024-01-30/
|   |   ├── test1.webm
|   |   └── test2.webm
|   └── 2024-01-31/
|
├── logs/
|   ├── 2024-01-30-execution.log
|   └── 2024-01-31-execution.log
|
└── test-data/
    ├── users.json
    └── products.csv
```

Uploading Files from EC2

Method 1: AWS CLI (Simple)

```
 # On EC2, after tests complete:

# Upload single file
aws s3 cp playwright-report/index.html   s3://test-automation-artifacts-yo

# Upload entire directory
aws s3 cp playwright-report/   s3://test-automation-artifacts-yourname/rep

# Sync (only upload changed files)
```

```
aws s3 cp report.html   s3://test-automation-artifacts-yourname/reports/
```

Method 2: Node.js SDK (Programmatic)

```
 # On EC2, install AWS SDK
npm install @aws-sdk/client-s3
```

Create `upload-results.js` :

```
 const { S3Client, PutObjectCommand } = require('@aws-sdk/client-s3');
const fs = require('fs');
const path = require('path');

const s3Client = new S3Client({ region: 'us-east-1' });
const bucketName = 'test-automation-artifacts-yourname';

async function uploadFile(filePath, s3Key) {
  const fileContent = fs.readFileSync(filePath);
  const contentType = getContentType(filePath);

  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: s3Key,
    Body: fileContent,
    ContentType: contentType,
    Metadata: {
      'test-run': process.env.BUILD_NUMBER || 'local',
      'environment': process.env.TEST_ENV || 'staging',
      'upload-time': new Date().toISOString()
    }
  });

  try {
    await s3Client.send(command);
    console.log(`✅ Uploaded: ${s3Key}`);
  } catch (error) {
    console.error(`❌ Failed: ${s3Key}`, error.message);
  }
}

async function uploadDirectory(dirPath, s3Prefix) {
  const files = getAllFiles(dirPath);
```

```
    const s3Key = `${s3Prefix}/${relativePath}`;
    await uploadFile(file, s3Key);
  }
}

function getAllFiles(dirPath, arrayOfFiles = []) {
  const files = fs.readdirSync(dirPath);

  files.forEach(file => {
    const filePath = path.join(dirPath, file);
    if (fs.statSync(filePath).isDirectory()) {
      arrayOfFiles = getAllFiles(filePath, arrayOfFiles);
    } else {
      arrayOfFiles.push(filePath);
    }
  });

  return arrayOfFiles;
}

function getContentType(filePath) {
  const ext = path.extname(filePath).toLowerCase();
  const types = {
    '.html': 'text/html',
    '.css': 'text/css',
    '.js': 'application/javascript',
    '.json': 'application/json',
    '.png': 'image/png',
    '.jpg': 'image/jpeg',
    '.webm': 'video/webm',
    '.log': 'text/plain'
  };
  return types[ext] || 'application/octet-stream';
}

// Usage
const date = new Date().toISOString().split('T')[0];
uploadDirectory('playwright-report', `reports/playwright/${date}`);
```

Run after tests:

```
 npx playwright test
node upload-results.js
```

```
 # Generate URL valid for 7 days
aws s3 presign   s3://test-automation-artifacts-yourname/reports/playwrigh

# Output:
https://test-automation-artifacts-yourname.s3.amazonaws.com/reports/...?X-
```

Share this URL with your team!

Lifecycle Policies (Auto-Cleanup)

Create `lifecycle.json` :

```
{
  "Rules": [
    {
      "Id": "DeleteOldReports",
      "Status": "Enabled",
      "Filter": {
        "Prefix": "reports/"
      },
      "Expiration": {
        "Days": 30
      }
    },
    {
      "Id": "ArchiveScreenshots",
      "Status": "Enabled",
      "Filter": {
        "Prefix": "screenshots/"
      },
      "Transitions": [
        {
          "Days": 7,
          "StorageClass": "GLACIER_INSTANT_RETRIEVAL"
        }
      ]
    },
    {
      "Id": "DeleteOldLogs",
      "Status": "Enabled",
```

```
      "Expiration": {
        "Days": 90
      }
    }
  ]
}
```

Apply lifecycle:

```
aws s3api put-bucket-lifecycle-configuration   --bucket test-automation-ar
```

What this does:

- Reports: Auto-delete after 30 days
- Screenshots: Move to cheaper storage after 7 days
- Logs: Delete after 90 days

Savings: ~60% on storage costs!

Lambda for Serverless Testing

What is Lambda?

Lambda = Run Code Without Servers

Traditional:

```
EC2: $0.01/hour × 24 hours × 30 days = $7.20/month
(Even if code runs 1 minute/day!)
```

Lambda:

```
Example: Health check running 1000 times/day
= 30,000 requests/month = $0.006/month
= Almost FREE!
```

Real-World Use Cases for QA

Perfect for Lambda:

- API health checks
- Smoke tests (critical flows)
- Data validation
- Scheduled lightweight tests
- WebHook testing

Not for Lambda:

- Full UI testing (15-min limit)
- Heavy browser automation
- Tests with dependencies
- Long-running tests

Creating Lambda Function

Example: API Health Check

Create `health-check.js` :

```
exports.handler = async (event) => {
  const apiEndpoint = process.env.API_ENDPOINT || 'https://api.example.com

  console.log(`🏥 Checking health of: ${apiEndpoint}`);

  try {
    const response = await fetch(apiEndpoint, {
      method: 'GET',
```

```
    const data = await response.json();

    if (response.ok && data.status === 'healthy') {
      console.log('✅ API is healthy');
      return {
        statusCode: 200,
        body: JSON.stringify({
          message: 'API is healthy',
          endpoint: apiEndpoint,
          responseTime: response.headers.get('X-Response-Time'),
          timestamp: new Date().toISOString()
        })
      };
    } else {
      console.error('❌ API returned unhealthy status');
      throw new Error(`Unhealthy status: ${data.status}`);
    }
  } catch (error) {
    console.error('❌ Health check failed:', error.message);

    // Send alert (SNS, email, Slack)
    await sendAlert(error.message);

    return {
      statusCode: 500,
      body: JSON.stringify({
        message: 'Health check failed',
        error: error.message,
        endpoint: apiEndpoint,
        timestamp: new Date().toISOString()
      })
    };
  }
};

async function sendAlert(message) {
  // TODO: Implement alert mechanism
  // Options: SNS, email, Slack webhook, PagerDuty
}
```

Deploy Lambda:

```
# Create IAM role for Lambda
aws iam create-role   --role-name lambda-health-check-role   --assume-role

# Attach basic execution policy
aws iam attach-role-policy   --role-name lambda-health-check-role   --poli

# Create Lambda function
aws lambda create-function   --function-name api-health-check   --runtime
```

Scheduling Lambda (EventBridge)

Run every 5 minutes:

```
 # Create EventBridge rule
aws events put-rule   --name health-check-every-5-min   --schedule-express

# Add Lambda as target
aws events put-targets   --rule health-check-every-5-min   --targets "Id"=

# Grant permission
aws lambda add-permission   --function-name api-health-check   --statement
```

Schedule Expressions:

```
 rate(5 minutes)          # Every 5 minutes
rate(1 hour)             # Every hour
rate(1 day)              # Daily

cron(0 9 * * ? *)        # Daily at 9 AM UTC
cron(0 18 * * ? *)       # Daily at 6 PM UTC
cron(0 9 ? * MON-FRI *)  # Weekdays at 9 AM
cron(0/10 * * * ? *)     # Every 10 minutes
```

Complete End-to-End Example

1. Tests on local machine
2. Deploy to EC2
3. Run tests daily
4. Upload results to S3
5. Health check with Lambda

Implementation:

Step 1: Create deployment script `deploy-to-aws.sh` :

```
 #!/bin/bash
set -e

EC2_HOST="ubuntu@YOUR_EC2_IP"
KEY_FILE="playwright-test-key.pem"
S3_BUCKET="test-automation-artifacts-yourname"
DATE=$(date +%Y-%m-%d)
TIMESTAMP=$(date +%Y%m%d-%H%M%S)

echo "🚀 Deploying tests to AWS EC2..."

# Package tests
echo "📦 Packaging tests..."
tar -czf tests.tar.gz   --exclude='node_modules'   --exclude='playwright-r

# Copy to EC2
echo "📤 Uploading to EC2..."
scp -i $KEY_FILE tests.tar.gz $EC2_HOST:~/test-automation/

# Deploy and run on EC2
echo "🏃 Running tests on EC2..."
ssh -i $KEY_FILE $EC2_HOST << 'ENDSSH'
  cd ~/test-automation
  tar -xzf tests.tar.gz
  npm install

  # Run tests
  echo "🧪 Executing test suite..."
  npm test
```

```
  TIMESTAMP=$(date +%Y%m%d-%H%M%S)

  aws s3 sync playwright-report/     s3://test-automation-artifacts-yourna

  # Generate and save report URL
  REPORT_URL=$(aws s3 presign     s3://test-automation-artifacts-yourname/

  echo "✅ Tests complete!"
  echo "📊 Report: $REPORT_URL"
ENDSSH

echo "✅ Deployment complete!"
```

Step 2: Schedule with cron

```
 # Edit crontab
crontab -e

# Add: Run daily at 9 AM
0 9 * * * /path/to/deploy-to-aws.sh >> ~/logs/aws-tests.log 2>&1
```

Step 3: Create GitHub Actions workflow `.github/workflows/aws-tests.yml` :

```
 name: AWS Test Execution

on:
  push:
    branches: [ main ]
  schedule:
    - cron: '0 9 * * *'  # Daily at 9 AM UTC

jobs:
  deploy-and-test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
```

```
    - name: Start EC2 instance
      run: |
        aws ec2 start-instances --instance-ids ${{ secrets.EC2_INSTANCE_ID
        aws ec2 wait instance-running --instance-ids ${{ secrets.EC2_INSTA

    - name: Get EC2 IP
      id: ec2-ip
      run: |
        IP=$(aws ec2 describe-instances           --instance-ids ${{ secre
        echo "ip=$IP" >> $GITHUB_OUTPUT

    - name: Deploy tests
      run: |
        # Create SSH key file
        echo "${{ secrets.EC2_SSH_KEY }}" > key.pem
        chmod 400 key.pem

        # Package and upload
        tar -czf tests.tar.gz .
        scp -i key.pem -o StrictHostKeyChecking=no           tests.tar.gz

    - name: Run tests
      run: |
        ssh -i key.pem -o StrictHostKeyChecking=no           ubuntu@${{ st
          cd ~/test-automation
          tar -xzf tests.tar.gz
          npm install
          npm test
        ENDSSH

    - name: Upload results to S3
      run: |
        ssh -i key.pem ubuntu@${{ steps.ec2-ip.outputs.ip }}           "cd

    - name: Generate report URL
      id: report-url
      run: |
        URL=$(aws s3 presign           s3://test-automation-artifacts-your
        echo "url=$URL" >> $GITHUB_OUTPUT

    - name: Comment PR with results
      if: github.event_name == 'pull_request'
      uses: actions/github-script@v7
      with:
        script: |
```

```
            repo: context.repo.repo,
            body: '🧪 Test Results: [${{ steps.report-url.outputs.url }}]'
          })

    - name: Stop EC2 instance
      if: always()
      run: |
        aws ec2 stop-instances --instance-ids ${{ secrets.EC2_INSTANCE_ID
```

Cost Optimization Strategies

Understanding Costs

Typical Monthly Costs:

```
 Scenario 1: Learning (Free Tier):
├── EC2 t2.micro: $0 (750 hours free)
├── S3: $0 (5 GB free)
├── Lambda: $0 (1M requests free)
└── Total: $0/month ✅

Scenario 2: Small Team:
├── EC2 t2.medium (8 hrs/day): $11
├── S3 (50 GB): $1.15
├── Lambda (100K requests): $0.02
├── Data transfer: $0.90
└── Total: ~$13/month

Scenario 3: Large Team:
├── EC2 t2.large (24/7): $69
├── EC2 spot instances (5x): $35
├── S3 (500 GB): $11.50
├── Lambda (1M requests): $0.20
├── Data transfer: $9
└── Total: ~$125/month

Traditional Setup: $500-1000/month ❌
```

```
 # Automate stop at night
0 19 * * * aws ec2 stop-instances --instance-ids i-xxxxx

# Auto-start in morning
0 8 * * * aws ec2 start-instances --instance-ids i-xxxxx

# Savings: ~70% ($69 → $20/month)
```

2. Use Spot Instances

```
 # Regular price: $0.094/hour
# Spot price: $0.028/hour (70% discount)

aws ec2 request-spot-instances   --spot-price "0.05"   --instance-count 1

# Risk: Can be terminated (rare, <5% chance)
# Best for: Non-critical test runs
```

3. S3 Lifecycle Policies

```
{
  "Rules": [{
    "Id": "OptimizeStorage",
    "Status": "Enabled",
    "Transitions": [
      {
        "Days": 7,
        "StorageClass": "INTELLIGENT_TIERING"
      },
      {
        "Days": 30,
        "StorageClass": "GLACIER_INSTANT_RETRIEVAL"
      }
    ],
    "Expiration": {
      "Days": 90
    }
  }]
}
```

4. Set Billing Alerts

```
 # Create SNS topic
aws sns create-topic --name billing-alerts

# Subscribe email
aws sns subscribe   --topic-arn arn:aws:sns:us-east-1:ACCOUNT:billing-aler

# Create billing alarm
aws cloudwatch put-metric-alarm   --alarm-name MonthlyBudget   --alarm-des
```

5. Right-Size Instances

```
 Too Small → Tests timeout, frustration
Too Large → Waste money

Finding the Right Size:
1. Start with t2.micro (free tier)
2. Monitor CPU/memory during test run
3. If >80% utilized → upgrade one size
4. If <30% utilized → downgrade

Tools: CloudWatch metrics, AWS Compute Optimizer
```

Monitoring Costs

AWS Cost Explorer:

```
 AWS Console → Cost Management → Cost Explorer

View:
- Daily costs by service
- Forecasted monthly bill
- Cost trends
- Recommendations
```

CLI Method:

```
# Output:
{
  "EC2": 15.23,
  "S3": 1.45,
  "Lambda": 0.02
}
```

Practice Exercises

Exercise 1: Setup Complete Environment (30 min)

1. Create AWS account
2. Enable MFA
3. Create IAM user
4. Install and configure AWS CLI
5. Verify access

Deliverable: Screenshot of `aws sts get-caller-identity`

Exercise 2: Launch and Configure EC2 (45 min)

1. Launch t2.micro instance
2. Create and configure security group
3. SSH into instance
4. Run setup script
5. Verify installations

Deliverable: Screenshot showing Node, Chrome, Playwright versions

Exercise 3: Deploy and Run Tests (60 min)

3. Run tests remotely
4. Capture results

Deliverable: Test execution log from EC2

Exercise 4: S3 Artifact Management (45 min)

1. Create S3 bucket
2. Upload test report
3. Set lifecycle policy
4. Generate presigned URL
5. Share with instructor

Deliverable: Working presigned URL to your report

Exercise 5: Lambda Health Check (60 min)

1. Create Lambda function
2. Test manually
3. Set up EventBridge schedule (every 5 min)
4. Monitor CloudWatch logs
5. Verify automated execution

Deliverable: CloudWatch logs showing scheduled executions

Troubleshooting Common Issues

Issue 1: Can't Connect to EC2

```
Error: Connection timed out
```

```
3. Wrong IP address
4. Key file permissions incorrect

Solutions:
# Check instance state
aws ec2 describe-instances --instance-ids i-xxxxx

# Verify security group
aws ec2 describe-security-groups --group-ids sg-xxxxx

# Check key permissions
chmod 400 key.pem

# Get current IP
curl ifconfig.me
```

Issue 2: Tests Fail on EC2 but Work Locally

```
 Common causes:
1. Browser dependencies missing
2. Different Node version
3. Network restrictions
4. Insufficient memory

Solutions:
# Install all Playwright dependencies
npx playwright install-deps

# Check Node version matches
node --version  # Should be 18+

# Check available memory
free -h

# Run headless
npm test -- --headed=false
```

Issue 3: S3 Upload Permission Denied

```
# Configure AWS CLI on EC2
aws configure

# Or assign IAM role to EC2
aws iam create-role...
aws iam attach-role-policy...
aws ec2 associate-iam-instance-profile...

# Verify bucket permissions
aws s3 ls s3://your-bucket/
```

Issue 4: Lambda Timeout

```
Error: Task timed out after 3.00 seconds

Solutions:
# Increase timeout (max 15 minutes)
aws lambda update-function-configuration   --function-name your-function

# Optimize code
# - Reduce wait times
# - Parallel requests
# - Cache results
```

Summary & Next Steps

Key Takeaways

✅ **Cloud Benefits for QA:**

- Scalability (1 to 1000 instances)
- Cost-efficiency (pay-per-use)
- Global reach (test worldwide)
- No maintenance overhead

- S3: Unlimited artifact storage
- Lambda: Serverless health checks
- CloudWatch: Monitoring and logging

✅ **Best Practices:**

1. Never use root account
2. Enable MFA always
3. Tag all resources
4. Set billing alerts
5. Stop unused instances
6. Use lifecycle policies
7. Monitor costs regularly

✅ **Cost Management:**

- Free tier: $0 for 12 months
- Typical cost: $13-125/month
- Traditional setup: $500-1000/month
- Savings: 85-95%!

What's Next?

Day 13: CI/CD & Test Reporting

- GitHub Actions with AWS
- Jenkins integration
- ReportPortal setup
- Automated notifications

Day 14: AI in Test Automation

- AI-powered locators
- Self-healing tests

Day 15: Framework Design & Capstone

- Complete framework architecture
- Best practices
- Final project

Additional Resources

AWS Free Tier Calculator

https://calculator.aws/

AWS Documentation

- EC2: https://docs.aws.amazon.com/ec2/
- S3: https://docs.aws.amazon.com/s3/
- Lambda: https://docs.aws.amazon.com/lambda/

Tutorials

- AWS Getting Started: https://aws.amazon.com/getting-started/
- EC2 for Beginners: https://aws.amazon.com/ec2/getting-started/

Community

- AWS Forums: https://forums.aws.amazon.com/
- Stack Overflow: [aws] tag
- Reddit: r/aws

End of Day 12 - AWS for Test Automation

- Deploy tests on EC2
- Store artifacts in S3
- Create Lambda health checks
- Optimize costs
- Integrate with CI/CD

Next: Day 13 - CI/CD & Test Reporting

← BrowserStack Integration

CI/CD & Test Reporting →