(DVN-2025-12-19-TJM) Creation of AWS S3 for all Apps [ALL].md

Metadata

- **App**: [ALL]
- **Type**: [TASK]
- **Issue #**: [Launch]
- **Created**: [2025-12-19]
- **By**: [TJM and Cursor Composer]
- **Status**: Pending

This documents how I created Amazon Web Service (AWS) Storage Level-3 (S3) Buckets for all of MicroCODE's Web Apps.

Who:

I created an Account email: timothy.mcguire@mcode.com password: (see OneNote on Tim's iPhone)

What:

Web-based storage of app and website accessible files, fixed, uploaded, and generated. My NITxPRO profs recommended AWS S3. Period.

When:

This is an immediate need, for launching Regatta-RC and LADDERS in January (next month).

Where:

We will be using AWS S3 under a MicroCODE paid account, billed to our corporate AMEX Card.

Why:

We need a consistent method to store public assets (icons, images, sounds, app generated reports, etc.) and private app related assets (avatars, file uploads, app generated reports)

How:

This is the **recommended S3 bucket architecture** with **names, types, and intent**, aligned with how AWS expects SaaS apps to scale.

Naming rules (important)

You're doing this right — keep it boring and obvious:

Rules

- Lowercase
- No dots (.)
- No environment baked in
- Ownership first, app second, scope last

Pattern

```
{product}-{scope}-assets
```

Environments (do NOT create separate buckets)

Use **prefixes**, not buckets:

```
/prod/ <-- blank for production
/beta/
/dev/
```

Example:

```
regattarc-private-assets/users/123/
regattarc-private-assets/beta/users/123/
regattarc-private-assets/dev/users/123/

ladders-public-assets/site/logo.png
ladders-public-assets/beta/site/logo.png
ladders-public-assets/dev/site/logo.png
```

This avoids:

- Credential sprawl
- Policy duplication
- CloudFront explosion

---

1. MicroCODE Core (mcode.com)

A. **Public, shared, static assets only**

Bucket name

```
{product}-{scope}-assets
```

`mcode-public-assets`

Type

- S3 Standard
- Public read (via CloudFront, not direct S3)
- CloudFront is a CDN for speed + caching

Purpose

- Logos
- Email images
- Brand assets
- Shared CSS/images used across *all* MicroCODE properties

Structure

```
/brand/
    logo.png
    email-logo.png
/apps/
    fonts/
    icons/
    images/
    sounds/
    videos/
```

Access

- ❌ No uploads from apps
- ☑ Read-only
- Served via CloudFront: `assets.mcode.com`

B. Private user assets

Bucket

```
{product}-{scope}-assets
```

`mcode-private-assets`

Type

- S3 Standard
- **Private only**
- No public access (only via signed URLs with User_id checks)

Purpose

- User-uploaded content
- Generated HTML reports
- App-internal files

Structure

```
/users/{user_id}/
    uploads/
    exports/
    reports/
    jobs/{job_id}/
    temp/
```

Access model

- IAM Role (server)
- Pre-signed URLs for users
- Optional lifecycle rules (auto-delete temp)

---

2. Regatta RC (regattarc.com)

Public + private (user-scoped)

A. Public assets

Bucket

```
{product}-{scope}-assets
```

`regattarc-public-assets`

Type

- S3 Standard
- Public read (CloudFront)
- CloudFront is a CDN for speed + caching

Purpose

- Marketing images
- Public app assets
- Emails, landing pages

Structure

```
/brand/
    logo.png
    email-logo.png
/app/
    fonts/
    icons/
    images/
    sounds/
    videos/
```

Access

- ❌ No uploads from apps
- ☑ Read-only
- Served via CloudFront: `assets.regattarc.com`

B. Private user assets

Bucket

```
{product}-{scope}-assets
```

`regattarc-private-assets`

Type

- S3 Standard
- **Private only**
- No public access (only via signed URLs with User_id checks)

Purpose

- User-uploaded content
- Generated HTML reports
- App-internal files

Structure

```
/users/{user_id}/
    uploads/
    exports/
    reports/
    jobs/{job_id}/
    temp/
```

Access model

- IAM Role (server)
- Pre-signed URLs for users
- Optional lifecycle rules (auto-delete temp)

---

3. LADDERS (ladders.mcode.com)

Public + private (user-scoped, worker-generated)

A. Public assets

Bucket

```
ladders-public-assets
```

Type

- S3 Standard
- Public read (CloudFront)

Purpose

- App UI assets
- Marketing pages
- Email images

Structure

```
/brand/
    logo.png
    email-logo.png
/app/
    fonts/
    icons/
    images/
    sounds/
    videos/
```

Access

- ❌ No uploads from apps
- ☑ Read-only
- Served via CloudFront: `assets.ladders.mcode.com`

B. Private user assets + worker output

Bucket

```
ladders-private-assets
```

Type

- S3 Standard
- **Private only**
- No public access (only via signed URLs with User_id checks)

Purpose

- User-uploaded content
- Generated HTML reports

- App-internal files

Structure

```
/users/{user_id}/
    uploads/
    exports/
    reports/
    jobs/{job_id}/
    temp/
```

Access model

- IAM Role (server)
- Pre-signed URLs for users
- Optional lifecycle rules (auto-delete temp)

---

4. Security Configuration

4.1 Encryption at Rest (SSE-KMS)

Decision: Use SSE-KMS with Customer-Managed Keys (CMK)

Pros:

- Full control over encryption keys
- Audit trail of key usage via CloudTrail
- Ability to rotate keys on schedule
- Compliance requirements (GDPR, HIPAA, etc.)
- Key access can be restricted via IAM policies
- Keys can be disabled/revoked independently of data

Cons:

- Additional cost (~$1/month per key + $0.03 per 10,000 requests)
- Slightly more complex setup
- Key rotation requires careful planning (see below)

Key Rotation Strategy:

1. **Create key aliases** (not direct key IDs) for easier rotation:
    - `alias/mcode-private-assets-key`
    - `alias/regattarc-private-assets-key`
    - `alias/ladders-private-assets-key`
2. **Rotation Process** (does NOT break access to older files):
    - AWS KMS automatically maintains old key versions
    - When you rotate, AWS creates a new key version

- Old files encrypted with old key versions remain accessible
- New files use the new key version
- Old key versions are retained until explicitly deleted (after retention period)

3. **Implementation:**

- Enable automatic key rotation (annual) OR manual rotation (quarterly recommended)
- Set key deletion window to 30 days (prevents accidental deletion)
- Use key aliases in bucket encryption configuration (not key ARNs)

Key Naming Convention:

```
{app}-{scope}-assets-key
```

Example: `regattarc-private-assets-key`

NOT using user_id as key identifier - Keys are per-bucket, not per-user, for:

- Simpler key management
- Better performance (fewer key lookups)
- Easier rotation
- Lower costs

4.2 Versioning

Enable versioning on ALL private buckets (required for LADDERS file comparison feature).

Configuration:

- Enable versioning at bucket creation
- Cannot be disabled once enabled (only suspended)
- Each object modification creates a new version
- Previous versions retained until explicitly deleted

Lifecycle Policy Integration:

- Keep current version indefinitely
- Move non-current versions to S3 Glacier after 90 days
- Permanently delete non-current versions after 1 year (or per app policy)

Cost Consideration:

- Each version counts toward storage costs
- Use lifecycle policies to manage old versions automatically

---

4.3 MFA Delete Protection

Enable MFA Delete on ALL private buckets to prevent accidental or malicious deletion.

Requirements:

- Root account MFA device must be enabled
- MFA device required to:
  - Change versioning state
  - Permanently delete object versions
  - Delete bucket

Implementation:

- Use AWS root account MFA device (hardware token or authenticator app)
- Cannot use IAM user MFA for bucket-level MFA delete
- Set at bucket creation (cannot be enabled later without disabling versioning first)

Note: This is a one-time setup per bucket and provides critical protection.

4.4 Access Logging

Enable S3 Server Access Logging for all buckets.

Log Storage:

- Create dedicated logging bucket: `mcode-s3-access-logs`
- Logs stored with prefix: `{bucket-name}/YYYY/MM/DD/`
- Access via S3 console or CloudTrail integration

CloudTrail Integration:

- CloudTrail logs API calls (who, what, when, where)
- S3 access logs show object-level access details
- Both can be queried via AWS CLI, SDK, or CloudWatch Insights

Accessing Logs in Admin Console:

1. CloudTrail events → S3 bucket → CloudWatch Logs
2. Create CloudWatch Insights queries for specific patterns
3. Build custom admin dashboard using CloudTrail API:

```
// Example: Get S3 API calls for last 24 hours
const cloudtrail = new CloudTrailClient({ region: "us-east-1" });
const events = await cloudtrail.lookupEvents({
  LookupAttributes: [
    {
      AttributeKey: "ResourceName",
      AttributeValue: "arn:aws:s3:::regattarc-private-assets",
    },
  ],
  StartTime: new Date(Date.now() - 86400000),
  EndTime: new Date(),
});
```

Cost: CloudTrail is free for first copy of management events. S3 access logs: ~$0.50 per GB stored.

4.5 Pre-Signed URLs

Configuration for Semi-Permanent Files:

Most uploads are semi-permanent (avatars, backups, reports), so use longer expiration times:

Default Expiration Times:

- **Upload URLs:** 1 hour (sufficient for upload completion)
- **Download URLs (user files):** 7 days (avatars, backups)
- **Download URLs (reports):** 30 days (generated reports)
- **Download URLs (temp files):** 1 hour (temporary access)

Implementation:

- Generate URLs server-side with user context validation
- Always validate `user_id` matches file path before URL generation
- Store expiration time in database for audit trail
- Regenerate URLs on-demand (don't cache pre-signed URLs)

Security:

- URLs are cryptographically signed - cannot be forged
- Expiration enforced by AWS
- Revocation: Delete object or wait for expiration

4.6 CORS Policies

Required for Apps Hosted Elsewhere

Public Buckets CORS Configuration:

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET", "HEAD"],
    "AllowedOrigins": [
      "https://regattarc.com",
      "https://www.regattarc.com",
      "https://ladders.mcode.com",
      "https://mcode.com",
      "https://*.mcode.com"
    ],
    "ExposeHeaders": ["ETag", "Content-Length", "Content-Type"],
    "MaxAgeSeconds": 3600
  }
]
```

Private Buckets CORS Configuration:

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "AllowedOrigins": [
      "https://regattarc.com",
      "https://www.regattarc.com",
      "https://ladders.mcode.com"
    ],
    "ExposeHeaders": ["ETag", "Content-Length", "Content-Type"],
    "MaxAgeSeconds": 3600
  }
]
```

Note: CORS is for browser requests. Server-to-S3 requests don't need CORS.

4.7 Bucket Policies

Public Bucket Policy (Example: `regattarc-public-assets`)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudFrontOACReadOnly",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::regattarc-public-assets/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn":
"arn:aws:cloudfront::ACCOUNT_ID:distribution/DISTRIBUTION_ID"
        }
      }
    },
    {
      "Sid": "DenyDirectS3Access",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::regattarc-public-assets/*",
      "Condition": {
        "StringNotEquals": {
          "AWS:SourceArn":
"arn:aws:cloudfront::ACCOUNT_ID:distribution/DISTRIBUTION_ID"
        }
      }
    }
  ]
}
```

Private Bucket Policy (Example: `regattarc-private-assets`)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPublicAccess",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::regattarc-private-assets",
        "arn:aws:s3:::regattarc-private-assets/*"
      ],
      "Condition": {
        "Bool": {
          "aws:ViaAWSService": "false"
        }
      }
    },
    {
      "Sid": "AllowIAMRoleAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::ACCOUNT_ID:role/regattarc-server-role"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::regattarc-private-assets",
        "arn:aws:s3:::regattarc-private-assets/*"
      ]
    }
  ]
}
```

4.8 IAM Credential Management Best Practices

Never Use Root Account for Operations

Implementation Process:

1. **Create IAM Users** (for console access only):
   - `mcode-admin` (with MFA required)
   - `mcode-developer` (with MFA required)
   - Attach policies: `ReadOnlyAccess` + specific S3 permissions
2. **Create IAM Roles** (for applications):
   - `mcode-server-role` (for mcode.com servers)
   - `regattarc-server-role` (for regattarc.com servers)
   - `ladders-server-role` (for ladders.mcode.com servers)
   - `ladders-worker-role` (for background workers)
3. **Use AWS Secrets Manager** for credentials:

- Store access keys (if needed for legacy systems)
- Rotate automatically every 90 days
- Access via IAM roles (no hardcoded keys)

4. **Enable MFA** for all IAM users with console access

5. **Rotate Access Keys**:

   - Create new key
   - Update application (via Secrets Manager)
   - Deactivate old key (wait 7 days)
   - Delete old key

6. **Use IAM Roles for EC2/ECS/Lambda**:

   - Attach role to compute resource
   - No access keys needed
   - Automatic credential rotation

Best Practice: Applications should use IAM roles, not access keys. Only use access keys for external services that cannot assume roles.

4.9 Public Access Block Configuration

Block All Public Access Settings:

Enable ALL four settings for ALL buckets:

- ☑ Block public access to buckets and objects granted through new access control lists (ACLs)
- ☑ Block public access to buckets and objects granted through any access control lists (ACLs)
- ☑ Block public access to buckets and objects granted through new public bucket or access point policies
- ☑ Block public and cross-account access to buckets and objects through any public bucket or access point policies

Exception for Public Buckets:

- Public buckets still have "Block Public Access" enabled
- Access is granted ONLY via CloudFront Origin Access Control (OAC)
- CloudFront OAC bypasses public access block (by design)
- Direct S3 URLs are blocked

Implementation:

- Set at bucket creation
- Can be modified later (but not recommended)
- Applies to bucket and all objects

4.10 Content Security

File Type Validation:

- Whitelist approach (only allow specific types)
- Validate MIME type server-side before upload
- Reject files with double extensions (e.g., `file.pdf.exe`)
- Scan file headers (not just extensions)

File Size Limits:

- Avatars: 5 MB max
- User uploads: 100 MB max (configurable per app)
- Generated reports: 50 MB max
- Temporary files: 10 MB max

Content-Disposition Headers:

- Set `Content-Disposition: attachment` for downloads
- Set `Content-Disposition: inline` for images/previews
- Include filename: `Content-Disposition: attachment; filename="report.pdf"`

Virus Scanning:

- **NOT applied to app-generated files** (performance requirement)
- App-generated files are trusted (created by our code)
- User uploads: Consider AWS GuardDuty or third-party scanning (optional, not required)
- Mark app-generated files with metadata: `x-amz-meta-source: app-generated`

Implementation:

```
// Example: Upload app-generated file (no virus scan)
await s3.putObject({
  Bucket: bucket,
  Key: key,
  Body: buffer,
  Metadata: {
    source: "app-generated",
    "generated-by": "regattarc-server",
    "user-id": userId,
  },
});
```

---

5. Extensibility

5.1 New App Onboarding Process

Checklist for Adding a New App:

1. **Bucket Creation:**
   - ☐ Create public bucket: `{appname}-public-assets`
   - ☐ Create private bucket: `{appname}-private-assets`

- [ ] Apply naming convention validation
- [ ] Set region (match primary app region)

2. **Security Configuration:**

- [ ] Enable SSE-KMS encryption (create/assign CMK)
- [ ] Enable versioning (private bucket only)
- [ ] Enable MFA delete (private bucket only)
- [ ] Enable access logging
- [ ] Configure bucket policies
- [ ] Block public access (all buckets)

3. **CloudFront Setup:**

- [ ] Create CloudFront distribution for public bucket
- [ ] Configure Origin Access Control (OAC)
- [ ] Set up custom domain: `assets.{appname}.com`
- [ ] Configure SSL certificate
- [ ] Set cache behaviors and TTLs

4. **IAM Setup:**

- [ ] Create IAM role: `{appname}-server-role`
- [ ] Create IAM policy with least-privilege access
- [ ] Attach policy to role
- [ ] Create worker role if needed: `{appname}-worker-role`

5. **Lifecycle Policies:**

- [ ] Configure lifecycle rules for temp file cleanup
- [ ] Configure versioning lifecycle (Glacier transition)
- [ ] Set up cost allocation tags

6. **Monitoring:**

- [ ] Create CloudWatch alarms for bucket metrics
- [ ] Set up cost alerts
- [ ] Configure S3 access logging

7. **Documentation:**

- [ ] Update this document with new app details
- [ ] Document bucket structure and access patterns
- [ ] Update environment variable documentation

8. **Code Integration:**

- [ ] Update `server/helper/s3.js` with new bucket names
- [ ] Update environment variables
- [ ] Test upload/download flows
- [ ] Verify CloudFront URLs work

Template for New App Section:

```
## X. {App Name} ({appdomain}.com)

### A. Public assets

**Bucket:** `{appname}-public-assets`
**CloudFront:** `assets.{appdomain}.com`
[Follow structure from Regatta RC example]

### B. Private assets

**Bucket:** `{appname}-private-assets`
[Follow structure from Regatta RC example]
```

5.2 Cost Optimization & Lifecycle Policies

Cost Avoidance Strategy:

Beta Launch Target: < $100/month (Detroit/Mid-West region only, small user group)

TL;DR - What is Glacier and why do I need it?

Glacier is AWS's ultra-low-cost archival storage ($0.004/GB/month vs $0.023/GB/month for S3 Standard). Files stored in Glacier:

- Cost 80% less than regular S3 storage
- Take 1-5 minutes to retrieve (not instant)
- Perfect for old file versions and backups you rarely access
- **For Beta:** You DON'T need Glacier yet - your storage will be small. Add it later when you have significant data.

Beta Phase Lifecycle Policy (Simplified - No Glacier):

```
{
  "Rules": [
    {
      "Id": "TempFileCleanup",
      "Status": "Enabled",
      "Prefix": "temp/",
      "Expiration": {
        "Days": 7
      }
    },
    {
      "Id": "OldVersionsCleanup",
      "Status": "Enabled",
      "NoncurrentVersionExpiration": {
        "NoncurrentDays": 30
      }
    },
    {
      "Id": "UserFilesLifecycle",
      "Status": "Enabled",
      "Prefix": "users/",
      "Expiration": {
```

```
          "Days": 365
        },
        "Filter": {
          "Tag": {
            "Key": "retention",
            "Value": "standard"
          }
        }
      },
      {
        "Id": "KeepForeverFiles",
        "Status": "Enabled",
        "Prefix": "users/",
        "Filter": {
          "Tag": {
            "Key": "retention",
            "Value": "forever"
          }
        }
      }
    ]
}
```

Production Lifecycle Policy Template (Private Buckets - Use After Beta):

```
{
  "Rules": [
    {
      "Id": "TempFileCleanup",
      "Status": "Enabled",
      "Prefix": "temp/",
      "Expiration": {
        "Days": 7
      }
    },
    {
      "Id": "OldVersionsToGlacier",
      "Status": "Enabled",
      "NoncurrentVersionExpiration": {
        "NoncurrentDays": 90
      },
      "NoncurrentVersionTransitions": [
        {
          "NoncurrentDays": 30,
          "StorageClass": "STANDARD_IA"
        },
        {
          "NoncurrentDays": 90,
          "StorageClass": "GLACIER"
        }
      ]
    },
    {
      "Id": "UserFilesLifecycle",
      "Status": "Enabled",
      "Prefix": "users/",
      "Transitions": [
        {
          "Days": 90,
          "StorageClass": "STANDARD_IA"
        },
        {
          "Days": 365,
          "StorageClass": "GLACIER"
        }
```

```
        ],
        "Expiration": {
          "Days": 2555
        },
        "Filter": {
          "Tag": {
            "Key": "retention",
            "Value": "standard"
          }
        }
      },
      {
        "Id": "KeepForeverFiles",
        "Status": "Enabled",
        "Prefix": "users/",
        "Filter": {
          "Tag": {
            "Key": "retention",
            "Value": "forever"
          }
        }
      }
    ]
  }
```

User Communication Policy (Beta):

> **File Retention Policy (Beta Phase):**
>
> - Files **not marked "save forever"** will be permanently deleted after **1 year** of inactivity
> - Files **marked "save forever"** are retained indefinitely (subject to account status)
> - Temporary files (`/temp/` folder) are deleted after **7 days**
> - Old file versions are kept for **30 days** (for LADDERS file comparison feature)

User Communication Policy (Production - After Beta):

> **File Retention Policy:**
>
> - Files **not marked "save forever"** will be automatically archived after **90 days** of inactivity
> - Archived files moved to low-cost storage (still accessible, may take a few minutes to retrieve)
> - Files **not marked "save forever"** will be permanently deleted after **7 years** of inactivity
> - Files **marked "save forever"** are retained indefinitely (subject to account status)
> - Temporary files (`/temp/` folder) are deleted after **7 days**

Implementation:

- Tag files on upload: `retention: standard` or `retention: forever`
- Users can change retention tag via UI
- Lifecycle policies automatically enforce rules

Cost Allocation Tags:

- `App`: {appname}
- `Environment`: prod/beta/dev
- `BucketType`: public/private
- `Owner`: {team-name}

Beta Phase Cost Estimate:

Assuming small beta user group (< 50 users) in Detroit area:

- **S3 Storage:** ~50GB × $0.023/GB = **$1.15/month**
- **S3 Requests:** ~100K PUT + 500K GET × $0.0004/1K = **$0.24/month**
- **CloudFront:** Free tier (1TB transfer, 10M requests) = **$0/month**
- **KMS:** 3 keys × $1/key + ~10K requests × $0.03/10K = **$3.03/month**
- **CloudWatch:** Free tier (10 alarms) = **$0/month**
- **Data Transfer:** Minimal (single region) = **$0/month**

Total Estimated Beta Cost: ~$4-5/month (well under $100 target)

Note: Actual costs will scale with usage. Monitor closely and adjust lifecycle policies if needed.

Hard Limits:

- Set S3 bucket size alerts (see Monitoring section)
- Set monthly cost budgets per app ($100 for beta, $500 for production)
- Automatic disable on budget breach (see Monitoring section)

5.3 Regional Configuration

Initial Launch: Detroit/Mid-West Only

Primary Region: `us-east-2` (Ohio) - Closest to Detroit, lowest latency for Mid-West users

Alternative: `us-east-1` (N. Virginia) - Lowest cost, most services available (slightly higher latency)

Regional Strategy:

1. **CloudFront Distribution:**
    - CloudFront automatically serves from nearest edge location
    - Even with single region, CloudFront provides caching and HTTPS
    - Reduces origin requests (saves costs)
    - **Beta:** Use CloudFront for public assets only (cost optimization)
2. **Single Region Setup:**
    - All buckets in `us-east-2` (Ohio) for beta launch
    - No cross-region replication needed initially
    - Simplifies setup and reduces costs
3. **Future Expansion (Post-Beta):**
    - When expanding beyond Mid-West, CloudFront will automatically optimize
    - Consider cross-region replication only if:
        - Disaster recovery requirements demand it
        - Compliance requires data residency in specific regions
        - User base grows significantly in other regions

Beta Phase Configuration:

- Region: `us-east-2` (Ohio) or `us-east-1` (Virginia)
- CloudFront: Enabled for public buckets only
- No cross-region replication
- Focus on cost optimization for small user base

Cost Impact:

- Single region: No data transfer costs
- CloudFront: Free tier includes 1TB data transfer out, 10M requests/month
- **Estimated Beta Cost:** < $50/month for storage + CloudFront (assuming < 100GB storage, < 1M requests)

5.4 Scaling Considerations

Request Rate Limits:

S3 supports **3,500 PUT/COPY/POST/DELETE requests per second per prefix** and **5,500 GET/HEAD requests per second per prefix**.

Prefix Distribution Strategy:

To avoid hitting limits, distribute objects across prefixes:

```
/users/{user_id}/uploads/{hash_prefix}/{filename}
```

Where `hash_prefix` is first 2 characters of MD5 hash of filename:

- Distributes load across 256 prefixes (00-ff)
- Prevents hot-spotting on single prefix
- No code changes needed (transparent to users)

CloudFront Cache Invalidation:

- Limit invalidations (costs $0.005 per invalidation after first 1,000/month)
- Use versioned URLs instead: `logo-v2.png` instead of invalidating `logo.png`
- Set appropriate cache TTLs (24 hours for static assets, 1 hour for dynamic)

Monitoring:

- CloudWatch metric: `NumberOfObjects` per prefix
- Alert on high request rates (> 2,000/sec per prefix)
- Automatically add more prefix levels if needed

5.5 Environment Management

Prefix-Based Environments:

Production uses no prefix, beta/dev use prefixes:

```
/prod/  → (blank, root level)
/beta/
/dev/
```

Promotion Process (dev → beta → prod):

1. **Copy objects** (not move, keep source):

   ```
   aws s3 sync s3://bucket/dev/users/ s3://bucket/beta/users/
   ```

2. **Update application config** to point to new prefix

3. **Verify** all objects accessible

4. **Delete old prefix** after verification period (30 days)

Environment-Specific IAM Policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:PutObject"],
      "Resource": "arn:aws:s3:::regattarc-private-assets/dev/*"
    },
    {
      "Effect": "Deny",
      "Action": "s3:DeleteObject",
      "Resource": "arn:aws:s3:::regattarc-private-assets/prod/*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalTag/Environment": "production-admin"
        }
      }
    }
  ]
}
```

Environment Tagging:

- Tag all objects with `Environment: prod/beta/dev`
- Use tags for lifecycle policies
- Use tags for cost allocation

---

6. Maintainability

6.1 Implementation Examples

IAM Policy Example (regattarc-render-app user)

Policy for Render.io hosted Regatta-RC application:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPublicBucketRead",
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:ListBucket"],
      "Resource": [
        "arn:aws:s3:::regattarc-public-assets",
        "arn:aws:s3:::regattarc-public-assets/*"
      ]
    },
    {
      "Sid": "AllowPrivateBucketFullAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket",
        "s3:GetObjectVersion",
        "s3:PutObjectTagging"
      ],
      "Resource": [
        "arn:aws:s3:::regattarc-private-assets",
        "arn:aws:s3:::regattarc-private-assets/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    },
    {
      "Sid": "AllowKMSKeyUsage",
      "Effect": "Allow",
      "Action": ["kms:Decrypt", "kms:GenerateDataKey"],
      "Resource": "arn:aws:kms:us-east-2:ACCOUNT_ID:key/KEY_ID"
    }
  ]
}
```

Attach this policy to IAM user: `regattarc-render-app`

Render.io Environment Variables:

- `AWS_ACCESS_KEY_ID` - From IAM user credentials
- `AWS_SECRET_ACCESS_KEY` - From IAM user credentials
- `AWS_REGION` - `us-east-2`

CloudFront Distribution Configuration

Origin Access Control (OAC) Setup:

1. Create OAC in CloudFront console

2. Attach to S3 bucket origin
3. Update bucket policy to allow OAC (see Bucket Policies section)

CloudFront Behavior Settings:

```
{
  "CachePolicyId": "4135ea2d-6df8-44a3-9df3-4b5a84be39ad", //
CachingOptimized
  "Compress": true,
  "ViewerProtocolPolicy": "redirect-to-https",
  "AllowedMethods": {
    "Quantity": 2,
    "Items": ["GET", "HEAD"]
  },
  "CachedMethods": {
    "Quantity": 2,
    "Items": ["GET", "HEAD"]
  }
}
```

Custom Error Responses:

- 403 → 404 (hide S3 structure)
- 404 → Custom error page

Lifecycle Policy Example (Complete)

See section 5.2 for full lifecycle policy JSON.

6.2 Monitoring & Alerting

CloudWatch Metrics to Monitor:

1. **Bucket Size:**
   - Metric: `BucketSizeBytes`
   - Alarm: > 1 TB per bucket
   - Action: Email + SNS notification
2. **Request Count:**
   - Metric: `NumberOfObjects`
   - Alarm: > 10 million objects
   - Action: Email notification
3. **4xx Errors:**
   - Metric: `4xxErrors`
   - Alarm: > 100 errors in 5 minutes
   - Action: Email + PagerDuty (if configured)
4. **5xx Errors:**

- Metric: `5xxErrors`
- Alarm: > 10 errors in 5 minutes
- Action: Email + PagerDuty

Cost Monitoring & Alerts:

Setup Process:

1. **Create Cost Budget:**

   Beta Phase Budget:

```
{
  "BudgetName": "S3-Beta-Monthly-Budget",
  "BudgetLimit": {
    "Amount": "50",
    "Unit": "USD"
  },
  "TimeUnit": "MONTHLY",
  "BudgetType": "COST",
  "CostFilters": {
    "Service": ["Amazon Simple Storage Service", "Amazon
CloudFront"]
  }
}
```

   Production Budget (Post-Beta):

```
{
  "BudgetName": "S3-Production-Monthly-Budget",
  "BudgetLimit": {
    "Amount": "100",
    "Unit": "USD"
  },
  "TimeUnit": "MONTHLY",
  "BudgetType": "COST",
  "CostFilters": {
    "Service": ["Amazon Simple Storage Service", "Amazon
CloudFront"]
  }
}
```

2. **Create Alerts at Different Thresholds:**

   Beta Phase Alerts:

   - **50% of budget ($50):** Email notification
   - **80% of budget ($80):** Email + SNS (escalate to team lead)
   - **100% of budget ($100):** Email + SNS + **AUTOMATIC DISABLE**
   - **120% of budget ($120):** Email + SNS + Alert AWS account admin

   Production Alerts:

   - **50% of budget:** Email notification
   - **80% of budget:** Email + SNS (escalate to team lead)

- **100% of budget:** Email + SNS + **AUTOMATIC DISABLE**
- **120% of budget:** Email + SNS + Alert AWS account admin

3. **Hard Disable Implementation:**

- Use AWS Lambda function triggered by CloudWatch alarm
- Lambda attaches deny-all policy to IAM roles
- Prevents new uploads (reads still work for existing files)
- Manual re-enable required after investigation

Lambda Function for Hard Disable:

```
// lambda/s3-budget-disable.js
const { IAMClient, PutUserPolicyCommand } = require("@aws-sdk/client-
iam");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

exports.handler = async (event) => {
  const iam = new IAMClient({ region: "us-east-2" });
  const sns = new SNSClient({ region: "us-east-2" });

  // Deny all S3 actions
  const denyPolicy = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Deny",
        Action: "s3:*",
        Resource: "*",
      },
    ],
  };

  // Apply to all Render.io app IAM users
  const users = [
    "mcode-render-app",
    "regattarc-render-app",
    "ladders-render-app",
    "ladders-render-worker",
  ];

  for (const userName of users) {
    await iam.send(
      new PutUserPolicyCommand({
        UserName: userName,
        PolicyName: "BudgetExceeded-DenyAll",
        PolicyDocument: JSON.stringify(denyPolicy),
      })
    );
  }

  // Send notification via SNS (emails handled by Mailgun, not AWS SES)
  await sns.send(
    new PublishCommand({
      TopicArn: process.env.SNS_TOPIC_ARN,
      Subject: "AWS S3 Budget Exceeded - Access Disabled",
      Message: `S3 budget exceeded. Access disabled for all Render.io
applications. Manual re-enable required.`,
    })
  );
};
```

Note: Since apps are hosted on Render.io (not AWS compute), we disable IAM user policies. Apps will fail gracefully when AWS access is denied. SNS notifications can trigger webhooks to Mailgun or other services for email alerts.

Cost Breakdown by App:

- Use cost allocation tags (see section 5.2)
- Create separate budgets per app
- Dashboard: AWS Cost Explorer filtered by tags

Costs Involved:

- CloudWatch alarms: **Free** (first 10 alarms)
- SNS notifications: **$0.50 per 100,000 notifications** (emails sent via Mailgun, not AWS SES)
- Lambda invocations: **Free tier: 1M requests/month**
- Cost Explorer: **Free**
- **Total monitoring cost: ~$1-2/month** (minimal SNS usage)

6.3 Backup Strategy

Customer Expectation: Zero Data Loss

Backup Approach:

1. **Versioning (Primary Protection):**
    - Enabled on all private buckets
    - Protects against accidental deletion/modification
    - Previous versions retained per lifecycle policy
2. **Cross-Region Replication (Disaster Recovery):**
    - Replicate to secondary region: `eu-west-1`
    - Protects against regional outages
    - RTO: < 1 hour (manual failover)
    - RPO: < 15 minutes (replication lag)
3. **Lifecycle Policy to Glacier:**
    - Non-current versions → Glacier after 90 days
    - Provides long-term archival
    - Cost-effective for compliance
4. **Point-in-Time Recovery (Optional):**
    - S3 Object Lock (Governance mode)
    - Prevents deletion for retention period
    - Additional cost: ~$0.01 per GB/month

Backup Verification:

- Monthly test restore from Glacier
- Verify cross-region replication working
- Document restore procedures

RTO/RPO Targets:

- **RTO (Recovery Time Objective):** 1 hour
- **RPO (Recovery Point Objective):** 15 minutes

Implementation:

- Enable versioning (already planned)
- Enable cross-region replication (see section 5.3)
- Configure lifecycle policies (see section 5.2)
- Document restore procedures in runbook

6.4 Code Integration

Update `server/helper/s3.js`

Required Changes:

1. **Multi-Bucket Support:**
   - Support bucket selection by app/scope
   - Environment prefix handling
   - CloudFront URL generation
2. **User-Scoped Paths:**
   - Generate paths: `/users/{user_id}/uploads/{filename}`
   - Validate user_id matches authenticated user
3. **SSE-KMS Integration:**
   - Add KMS key parameter
   - Set encryption headers on upload
4. **Metadata Support:**
   - Support file metadata (retention tags, source, etc.)
   - Set Content-Disposition headers

Updated Helper Structure:

```
// server/helper/s3.js (conceptual update)
const {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
} = require("@aws-sdk/client-s3");
const { getSignedUrl } = require("@aws-sdk/s3-request-presigner");
```

```
const getBucket = (app, scope, environment = "prod") => {
  const envPrefix = environment === "prod" ? "" : `${environment}/`;
  return {
    name: `${app}-${scope}-assets`,
    prefix: envPrefix,
    kmsKeyId:

process.env[`KMS_${app.toUpperCase()}_${scope.toUpperCase()}_KEY_ID`],
  };
};

exports.upload = async function ({
  app,
  scope,
  userId,
  file,
  metadata = {},
  environment = "prod",
}) {
  const bucket = getBucket(app, scope, environment);
  const key =
`${bucket.prefix}users/${userId}/uploads/${file.originalname}`;

  await s3.send(
    new PutObjectCommand({
      Bucket: bucket.name,
      Key: key,
      Body: buffer,
      ContentType: file.mimetype,
      ServerSideEncryption: "aws:kms",
      SSEKMSKeyId: bucket.kmsKeyId,
      Metadata: {
        "user-id": userId,
        source: "user-upload",
        retention: metadata.retention || "standard",
        ...metadata,
      },
    })
  );

  // Return CloudFront URL for public, signed URL for private
  if (scope === "public") {
    return `https://assets.${app}.com/${key}`;
  } else {
    return await exports.signedURL({ app, scope, key, environment });
  }
};
```

Update `admin/helper/s3.js`

Create new helper for admin console:

- Similar structure to `server/helper/s3.js`
- Additional methods for admin operations:
  - List all users' files
  - Delete user files (with audit log)
  - Generate admin access URLs
  - View access logs

Environment Variables

Required `.env` Variables:

```
# AWS Configuration
AWS_REGION=us-east-2  # us-east-2 (Ohio) for Detroit/Mid-West, or us-east-1 (Virginia) for lowest cost
AWS_ACCOUNT_ID=123456789012

# S3 Configuration
S3_REGION=us-east-2  # Match AWS_REGION - us-east-2 (Ohio) recommended for beta launch

# S3 Buckets (Public)
S3_MCODE_PUBLIC_BUCKET=mcode-public-assets
S3_REGATTARC_PUBLIC_BUCKET=regattarc-public-assets
S3_LADDERS_PUBLIC_BUCKET=ladders-public-assets

# S3 Buckets (Private)
S3_MCODE_PRIVATE_BUCKET=mcode-private-assets
S3_REGATTARC_PRIVATE_BUCKET=regattarc-private-assets
S3_LADDERS_PRIVATE_BUCKET=ladders-private-assets

# KMS Keys
KMS_MCODE_PRIVATE_KEY_ID=alias/mcode-private-assets-key
KMS_REGATTARC_PRIVATE_KEY_ID=alias/regattarc-private-assets-key
KMS_LADDERS_PRIVATE_KEY_ID=alias/ladders-private-assets-key

# CloudFront Distributions
CLOUDFRONT_MCODE_DOMAIN=assets.mcode.com
CLOUDFRONT_REGATTARC_DOMAIN=assets.regattarc.com
CLOUDFRONT_LADDERS_DOMAIN=assets.ladders.mcode.com

# AWS Credentials (for Render.io hosted apps)
# Store these in Render.io environment variables, NOT in .env files
AWS_ACCESS_KEY_ID=AKIA...  # From IAM user: mcode-render-app
AWS_SECRET_ACCESS_KEY=...   # From IAM user: mcode-render-app
# Note: Each Render.io service should use its own IAM user credentials

# Environment
NODE_ENV=production
S3_ENVIRONMENT=prod  # prod, beta, dev

# Monitoring
CLOUDWATCH_LOG_GROUP=/aws/s3/access-logs
COST_BUDGET_ALERT_SNS_ARN=arn:aws:sns:us-east-1:ACCOUNT_ID:s3-budget-alerts
```

Per-App Override Pattern:

- Apps can override with app-specific env vars
- Fall back to defaults if not set
- Document in app-specific README

6.5 Tagging Strategy

Cost Allocation Tags (Required):

Applied to all buckets and objects:

- `App`: `mcode` | `regattarc` | `ladders`

- `Environment`: `prod` | `beta` | `dev`
- `BucketType`: `public` | `private`
- `Owner`: `platform-team`
- `CostCenter`: `engineering`

Operational Tags:

- `Retention`: `standard` | `forever` | `temp`
- `Source`: `app-generated` | `user-upload`
- `CreatedBy`: `{service-name}`
- `LastModified`: `{timestamp}`

Tag-Based Access Policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:DeleteObject",
      "Resource": "arn:aws:s3:::bucket/*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/Retention": "temp"
        }
      }
    }
  ]
}
```

Implementation:

- Set tags on upload (via metadata or PutObjectTagging)
- Use tags in lifecycle policies
- Use tags for cost allocation reports

---

7. IAM Users & Policies (Render.io Hosting)

7.1 IAM User Structure

Create **one IAM user per app/service** for Render.io hosted applications:

- `mcode-render-app` (for mcode.com on Render.io)
- `regattarc-render-app` (for regattarc.com on Render.io)
- `ladders-render-app` (for ladders.mcode.com on Render.io)
- `ladders-render-worker` (for background workers on Render.io)

Each IAM user gets least-privilege access to its buckets only.

Access Key Management:

- Generate access keys for each IAM user

- Store in Render.io environment variables (encrypted at rest)
- Never commit keys to git or code repositories
- Rotate every 90 days

7.2 IAM Policy Examples

See section 6.1 for complete IAM policy JSON examples. Attach these policies to IAM users (not roles) for Render.io applications.

---

8. Implementation Checklist

Phase 1: Infrastructure Setup

1. **Create KMS Keys:**
    - ☐ Create CMK for each private bucket
    - ☐ Enable automatic rotation
    - ☐ Set key deletion window (30 days)
    - ☐ Document key ARNs and aliases
2. **Create Buckets:**
    - ☐ Create all 6 buckets with correct names
    - ☐ Enable versioning (private buckets only)
    - ☐ Enable MFA delete (private buckets only)
    - ☐ Enable SSE-KMS encryption (private buckets)
    - ☐ Block public access (all buckets)
    - ☐ Enable access logging
    - ☐ Configure bucket policies
    - ☐ Set CORS policies
3. **CloudFront Setup:**
    - ☐ Create distributions for public buckets (3)
    - ☐ Configure Origin Access Control (OAC)
    - ☐ Set up custom domains and SSL certificates
    - ☐ Configure cache behaviors
    - ☐ Test CloudFront URLs
4. **IAM Setup (Render.io Hosting):**
    - ☐ Create IAM users (4 users: mcode-render-app, regattarc-render-app, ladders-render-app, ladders-render-worker)
    - ☐ Create and attach IAM policies to users
    - ☐ Generate access keys for each IAM user
    - ☐ Store access keys in Render.io environment variables (encrypted)
    - ☐ Test S3 access from Render.io applications
    - ☐ Remove/rotate any old access keys

Phase 2: Lifecycle & Cost Management

5. **Lifecycle Policies:**
    - [ ] Create lifecycle policies for all buckets
    - [ ] Configure temp file cleanup
    - [ ] Configure versioning transitions
    - [ ] Test lifecycle rules
6. **Cost Management:**
    - [ ] Set up cost allocation tags
    - [ ] Create cost budgets
    - [ ] Configure CloudWatch alarms
    - [ ] Set up SNS notifications
    - [ ] Create Lambda for hard disable
    - [ ] Test budget alerts

Phase 3: Monitoring & Backup

7. **Monitoring:**
    - [ ] Enable CloudTrail
    - [ ] Configure CloudWatch metrics
    - [ ] Set up dashboards
    - [ ] Test alerting
8. **Backup:**
    - [ ] Enable cross-region replication (if needed)
    - [ ] Configure Glacier transitions
    - [ ] Document restore procedures
    - [ ] Test backup restore

Phase 4: Code Integration

9. **Code Updates:**
    - [ ] Update `server/helper/s3.js`
    - [ ] Create `admin/helper/s3.js`
    - [ ] Configure Render.io environment variables (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_REGION)
    - [ ] Update all S3 calls in codebase
    - [ ] Test upload/download flows from Render.io
    - [ ] Test CloudFront URLs
    - [ ] Test pre-signed URLs
    - [ ] Verify Mailgun integration for email notifications (separate from AWS)
10. **Documentation:**
    - [ ] Update developer documentation

- [ ] Create runbook for operations
- [ ] Document restore procedures
- [ ] Update API documentation

---

9. Next Steps

Immediate Actions (This Week)

1. **Review and approve this plan**
2. **Create AWS account structure** (if not done)
3. **Set up MFA on root account**
4. **Create KMS keys** (3 keys for private buckets)

Short Term (Next 2 Weeks)

1. **Create all 6 buckets** with security configurations
2. **Set up CloudFront distributions** (3 distributions)
3. **Create IAM users and policies** (for Render.io hosting)
4. **Generate access keys and configure Render.io environment variables**
5. **Configure lifecycle policies**
6. **Set up monitoring and alerts**

Medium Term (Next Month)

1. **Update codebase** (`server/helper/s3.js`, `admin/helper/s3.js`)
2. **Configure Render.io environment variables** (AWS credentials)
3. **Test end-to-end flows** from Render.io hosted apps
4. **Migrate existing files** (if any)
5. **Train team on new S3 structure and Render.io integration**

Long Term (Ongoing)

1. **Monitor costs and optimize**
2. **Review and adjust lifecycle policies**
3. **Add new apps using onboarding checklist**
4. **Regular backup verification**
5. **Security audits and key rotation**

Notes

Key Decisions Made:

- SSE-KMS with customer-managed keys (not user_id-based)
- Versioning enabled for all private buckets
- MFA delete enabled for all private buckets
- Prefix-based environments (not separate buckets)
- CloudFront for all public assets (not direct S3)
- Lifecycle policies for cost optimization

- Hard disable on budget breach
- Cross-region replication for disaster recovery (optional)
- No virus scanning for app-generated files (performance)
- **Apps hosted on Render.io** (not AWS compute) - using IAM users with access keys
- **Email handled by Mailgun** (not AWS SES) - SNS notifications can trigger webhooks to Mailgun

Open Questions:

- Cross-region replication: Enable immediately or wait?
- Glacier transition timing: 90 days or longer?
- Cost budget threshold: Start at $500/month and adjust?

Render.io Cost Management

Important: "Users" for Render.io Cost = Concurrent Active Users

Render.io costs are driven by **concurrent users** (active at the same time), not total registered users. This determines CPU/RAM needs and plan tier selection.

User Metrics Clarification:

- **Concurrent Users:** Users actively using the app simultaneously (drives Render.io plan tier)
- **Total Registered Users:** All users with accounts (affects database, not Render.io)
- **Daily Active Users (DAU):** Users who log in per day (affects bandwidth/requests)
- **Peak Concurrent:** Maximum simultaneous users (determines plan tier)

Rule of Thumb: `Peak Concurrent ≈ (DAU × 0.1 to 0.2)`

Beta Phase Cost Breakdown

Realistic Beta Usage Estimates:

- Total Registered Users: ~200-500
- Daily Active Users (DAU): ~50-100
- Peak Concurrent Users: ~10-20 users
- Average Concurrent Users: ~5-10 users

Render.io Service Configuration:

```
Web Service (Starter Plus - 1 GB RAM, 1 CPU):  $15/month
  └─ Handles peak concurrent load (10-20 users)

Onboarding Worker (Starter - 512 MB RAM):      $7/month
  └─ Processes email sends, trial expirations

Usage Worker (Starter - 512 MB RAM):           $7/month
  └─ Processes Stripe usage reports
──────────────────────────────────────────────────────
Render.io Subtotal:                            $29/month
```

Combined Infrastructure Costs (Beta):

```
Render.io (3 services):                          $29/month
AWS S3 (storage + requests):                     $4-5/month
CloudFront (public assets):                       $0/month (free tier)
KMS (encryption keys):                           $3/month
CloudWatch (monitoring):                         $0/month (free tier)
──────────────────────────────────────────────────────────
Total Infrastructure:                             $36-37/month
```

Well under $100/month target ☑

Plan Tier Selection Guide

Based on Peak Concurrent Users:

Peak Concurrent Users	Recommended Plan	RAM	CPU	Cost/Month
1-5 users	Starter	512 MB	0.5	$7
5-15 users	Starter Plus	1 GB	1	$15
15-50 users	Standard	2 GB	1	$25
50-100 users	Standard Plus	3 GB	1.5	$50
100+ users	Pro	4 GB	2	$80

Your Architecture Advantages:

1. **Background Workers:** Reduce web server load → lower plan tier needed
2. **Redis Queues (Bull):** Jobs processed asynchronously → web server stays responsive
3. **CloudFront:** Static assets served from CDN → reduces bandwidth costs

---

Cost Drivers (Detailed)

1. Service Count (Biggest Factor)

- Each service billed separately
- Your setup: 3 services (web + 2 workers)
- **Cost:** $7-25 per service depending on plan

2. Plan Tier (RAM/CPU Allocation)

- Determined by peak concurrent users
- Higher tier = more RAM/CPU = higher cost
- **Optimization:** Start low, monitor, upgrade only if needed

3. Bandwidth Usage

- Included: Varies by plan (Starter: ~100 GB, Starter Plus: ~200 GB)
- Overage: **$15 per 100 GB** (reduced from $30 in 2025)
- **Your apps:** Low for beta (< 50 users) → likely $0/month

- **Optimization:** CloudFront reduces origin bandwidth

4. Disk Storage

- Included: Usually 10-20 GB per plan
- Overage: ~$0.25/GB/month
- **Your apps:** Low storage needs → $0/month

5. Build Minutes

- Usually sufficient for small teams
- Overage: ~$0.003/minute
- **Your apps:** Minimal → $0/month

6. Database Services (If Using Render Managed DB)

- PostgreSQL: Starting at $95/month
- Redis: Starting at $32/month
- **Your setup:** External MongoDB/SQL → $0/month ☑

Cost Scaling Projections

User Count	Peak Concurrent	Web Plan	Workers	Monthly Cost
Beta (< 50 DAU)	10-20	Starter Plus	2 × Starter	$29
Small (100-500 DAU)	20-50	Standard	2 × Starter Plus	$55
Medium (500-2K DAU)	50-100	Standard Plus	2 × Standard	$125
Large (2K+ DAU)	100+	Pro	2 × Standard Plus	$230

Note: Costs scale with concurrent load, not total users. Background workers help keep web server costs lower.

Regional Deployment

Recommendation: Deploy Render.io services in `us-east-1` or `us-east-2` to match S3 region (`us-east-2`)

Benefits:

- Lower latency to S3 buckets
- Reduced data transfer costs
- Better performance for Detroit/Mid-West users

Configuration:

- Set region in Render.io service settings
- All services should use same region
- Match AWS_REGION environment variable

Monitoring & Optimization

Key Metrics to Track:

1. **CPU Usage:**
    - If consistently > 70% → Consider upgrading plan
    - Monitor in Render.io dashboard
    - Set alert at 70% threshold
2. **Memory Usage:**
    - If > 80% of allocated RAM → Consider upgrading plan
    - Monitor in Render.io dashboard
    - Set alert at 80% threshold
3. **Request Rate:**
    - Monitor requests per second
    - Affects bandwidth usage
    - Use CloudFront to reduce origin requests
4. **Response Time:**
    - If slow (> 500ms average) → May need more CPU/RAM
    - Monitor in Render.io metrics
5. **Bandwidth Usage:**
    - Track monthly bandwidth consumption
    - Set alert at 80% of included limit
    - CloudFront caching reduces origin bandwidth

Optimization Strategies:

1. **Combine Workers** (if possible):
    - Current: 2 separate worker services
    - Option: Single worker service running both jobs
    - **Savings:** $7/month (1 service instead of 2)
2. **Right-Size Plans:**
    - Start with Starter/Starter Plus
    - Monitor CPU/RAM usage monthly
    - Upgrade only if consistently > 70% CPU or > 80% RAM
3. **Use CloudFront:**
    - Serves static assets from CDN
    - Reduces Render.io bandwidth usage
    - Improves performance globally

4. **Free Tier for Dev/Staging:**
    - Use free tier for non-production environments
    - Production requires paid plans
5. **Monitor and Adjust:**
    - Review costs monthly
    - Adjust plans based on actual usage
    - Don't over-provision (waste money)

Cost Alerts & Budgets

Render.io Budget Recommendations:

Beta Phase:

- Set budget alert at $50/month (172% of estimated cost)
- Monitor weekly during first month
- Adjust based on actual usage

Production Phase:

- Set budget alert at $100/month initially
- Review and adjust based on growth
- Consider separate budgets per service

Combined AWS + Render.io Budget:

Beta Phase:

- AWS S3: $100/month budget (see section 6.2)
- Render.io: $50/month budget
- **Total Infrastructure Budget:** $150/month
- **Actual Estimated Cost:** $36-37/month ☑

Production Phase:

- AWS S3: $500/month budget
- Render.io: $200/month budget (scales with users)
- **Total Infrastructure Budget:** $700/month

Migration Considerations

When to Consider Alternatives:

Render.io is Good For:

- ☑ Beta launch (< 100 concurrent users)
- ☑ Small to medium scale (< 500 concurrent users)

- ☑ Teams wanting simplicity over control
- ☑ Cost: ~$29-125/month range

Consider Alternatives If:

- Costs exceed $200/month consistently
- Need more control over infrastructure
- Require specific AWS services integration
- Have DevOps capacity for Kubernetes

Alternative Options:

Platform	3 Services (Beta)	Notes
Render.io	$29/month	Current choice - easy setup
Railway.app	$20/month	Similar to Render.io
Fly.io	$10-20/month	Global edge, different model
AWS ECS/Fargate	~$66/month	More complex, more control
Heroku	$75/month	More expensive

Note on ECS Pricing: The $66/month includes Fargate compute (~$40) + Application Load Balancer (~$16) + ECR (~$1) + CloudWatch (~$2) + other AWS services. Render.io includes load balancing and basic logging in their price, while ECS requires separate AWS services. The $15-30/month estimate shown earlier was just Fargate compute cost, not the total infrastructure cost.

Migration Path:

- Start with Render.io for beta
- Monitor costs and performance
- Evaluate alternatives if costs exceed $200/month
- Keep architecture portable (Express/Node.js) for easy migration

---

Integration with AWS S3

How Render.io and AWS S3 Work Together:

1. **Render.io Web Service:**
    - Generates pre-signed URLs for S3 uploads
    - Serves CloudFront URLs for public assets
    - Handles user authentication before S3 access
2. **Render.io Workers:**
    - Process background jobs (onboarding, usage)
    - May generate files and upload to S3
    - Use same IAM user credentials as web service

3. **CloudFront:**
    - Serves public S3 assets globally
    - Reduces Render.io bandwidth usage
    - Improves performance for users

Cost Synergy:

- CloudFront reduces Render.io bandwidth costs
- S3 handles storage (cheaper than Render.io disk)
- Render.io handles compute (simpler than AWS ECS)

Environment Variables (Render.io):

```
# AWS Credentials (stored in Render.io dashboard)
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=...
AWS_REGION=us-east-2

# S3 Buckets
S3_REGATTARC_PUBLIC_BUCKET=regattarc-public-assets
S3_REGATTARC_PRIVATE_BUCKET=regattarc-private-assets

# KMS Keys
KMS_REGATTARC_PRIVATE_KEY_ID=alias/regattarc-private-assets-key

# CloudFront
CLOUDFRONT_REGATTARC_DOMAIN=assets.regattarc.com
```

Summary

Beta Phase (Target: < $100/month total infrastructure):

- Render.io: **$29/month** (3 services)
- AWS S3: **$4-5/month** (storage + requests)
- CloudFront: **$0/month** (free tier)
- KMS: **$3/month** (encryption keys)
- **Total: $36-37/month** ☑ Well under target

Key Takeaways:

1. **Concurrent users** drive Render.io costs (not total users)
2. **Service count** is biggest cost factor (each service billed separately)
3. **Background workers** reduce web server load → lower costs
4. **CloudFront** reduces bandwidth costs
5. **Monitor CPU/RAM** to right-size plans
6. **Start low, scale up** based on actual usage

Next Steps:

1. Deploy to Render.io with Starter Plus web + Starter workers
2. Monitor CPU/RAM usage for first month

3. Adjust plans based on actual usage
4. Set up cost alerts at $50/month for beta
5. Review costs monthly and optimize