Contents

RADIANT Data Retention Policy

Overview

This document defines data retention periods and deletion procedures for all data stored in the RADIANT platform.

Retention Schedule

User Data

Data Type	Active Retention	Archive	Total Retention	Deletion
Account info	Active + 30 days	N/A	Account lifetime + 30 days	Automatic
Usage history	2 years	5 years	7 years	Automatic
Chat history	90 days	1 year	1 year	Automatic
Uploaded files	Active	30 days post-delete	Active + 30 days	On request
API keys	Active	N/A	Revoked + 90 days	Automatic

System Data

Data Type	Retention	Purpose	Deletion
Audit logs	7 years	Compliance	Automatic
Access logs	2 years	Security	Automatic
Error logs	90 days	Debugging	Automatic
Metrics	15 months	CloudWatch default	Automatic
Backups	35 days	Recovery	Automatic

Billing Data

Data Type	Retention	Purpose	Legal Basis
Invoices	7 years	Tax compliance	Legal requirement
Transactions	7 years	Financial audit	Legal requirement
Payment methods	Active	Processing	Contract
Receipts	7 years	Tax compliance	Legal requirement

Implementation

Database Retention

```
-- Automatic data cleanup job (runs daily)
CREATE OR REPLACE FUNCTION cleanup_expired_data()
RETURNS void AS $$
BEGIN
  -- Delete expired chat messages (90 days)
  DELETE FROM chat_messages
  WHERE created_at < NOW() - INTERVAL '90 days'
  AND archived = false;

  -- Archive chat messages older than 90 days
  UPDATE chat_messages
  SET archived = true, archived_at = NOW()
  WHERE created_at < NOW() - INTERVAL '90 days'
  AND archived = false;

  -- Delete archived messages older than 1 year
  DELETE FROM chat_messages
  WHERE archived = true
  AND archived_at < NOW() - INTERVAL '1 year';

  -- Delete revoked API keys (90 days after revocation)
  DELETE FROM api_keys
  WHERE revoked_at < NOW() - INTERVAL '90 days';

  -- Delete expired sessions
```

```
  DELETE FROM user_sessions
  WHERE expires_at < NOW();

  -- Log cleanup
  INSERT INTO system_jobs (job_name, completed_at, records_affected)
  VALUES ('cleanup_expired_data', NOW(),
    (SELECT count(*) FROM pg_stat_user_tables WHERE relname IN
      ('chat_messages', 'api_keys', 'user_sessions')));
END;
$$ LANGUAGE plpgsql;

-- Schedule daily at 3 AM UTC
SELECT cron.schedule('data-cleanup', '0 3 * * *', 'SELECT cleanup_expired_data()');
```

S3 Lifecycle Policies

```
const bucket = new s3.Bucket(this, 'Storage', {
  lifecycleRules: [
    // User uploads - delete 30 days after object deletion marker
    {
      id: 'delete-old-versions',
      noncurrentVersionExpiration: cdk.Duration.days(30),
    },

    // Temp files - delete after 7 days
    {
      id: 'cleanup-temp',
      prefix: 'temp/',
      expiration: cdk.Duration.days(7),
    },

    // Logs - transition to Glacier after 90 days, delete after 2 years
    {
      id: 'archive-logs',
      prefix: 'logs/',
      transitions: [
        {
          storageClass: s3.StorageClass.GLACIER,
          transitionAfter: cdk.Duration.days(90),
        },
      ],
      expiration: cdk.Duration.days(730), // 2 years
    },

    // Backups - delete after 35 days
    {
      id: 'cleanup-backups',
      prefix: 'backups/',
```

```
      expiration: cdk.Duration.days(35),
    },
  ],
});
```

CloudWatch Log Retention

```
// Set retention for all log groups
const logRetention: Record<string, logs.RetentionDays> = {
  // Application logs
  '/aws/lambda/radiant-*': logs.RetentionDays.THREE_MONTHS,

  // API Gateway logs
  '/aws/apigateway/radiant-*': logs.RetentionDays.THREE_MONTHS,

  // Database logs (longer for compliance)
  '/aws/rds/cluster/radiant-*': logs.RetentionDays.TWO_YEARS,

  // Audit logs (longest retention)
  '/radiant/audit/*': logs.RetentionDays.TEN_YEARS,
};
```

Data Deletion

User-Initiated Deletion

Account Deletion Flow

```
async function deleteUserAccount(userId: string): Promise<void> {
  // 1. Verify identity (MFA required)
  await verifyIdentity(userId);

  // 2. Cancel active subscriptions
  await cancelSubscriptions(userId);

  // 3. Export data (optional, user-requested)
  const exportUrl = await exportUserData(userId);

  // 4. Mark account for deletion (30-day grace period)
  await markForDeletion(userId, {
    scheduledAt: addDays(new Date(), 30),
    reason: 'user_requested',
  });

  // 5. Send confirmation email
  await sendDeletionConfirmation(userId, exportUrl);
}

// Actual deletion after grace period
```

```typescript
async function executeAccountDeletion(userId: string): Promise<void> {
  // Delete in order (respect foreign keys)
  await deleteApiKeys(userId);
  await deleteChatHistory(userId);
  await deleteFiles(userId);
  await deletePreferences(userId);
  await deleteBillingHistory(userId); // Anonymize, don't delete
  await deleteAccount(userId);

  // Anonymize audit logs
  await anonymizeAuditLogs(userId);

  // Log deletion for compliance
  await logAccountDeletion(userId);
}
```

Data Categories Deleted

Category	Action	Timing
Profile	Delete	Immediate
Preferences	Delete	Immediate
Chat history	Delete	Immediate
Files	Delete	Immediate
API keys	Revoke + Delete	Immediate
Billing history	Anonymize	Immediate
Audit logs	Anonymize	Immediate
Backups	Excluded	Expires naturally

Administrative Deletion

```typescript
// Bulk deletion for compliance (e.g., GDPR request)
async function adminBulkDelete(
  tenantId: string,
  options: {
    dataTypes: string[];
    olderThan: Date;
    reason: string;
    approvedBy: string[];
  }
): Promise<DeletionReport> {
  // Require dual admin approval
  if (options.approvedBy.length < 2) {
    throw new Error('Dual admin approval required');
  }

  // Log the deletion request
  await logAdminAction({
```

```
    action: 'bulk_delete',
    tenantId,
    options,
  });

  // Execute deletion
  const results = await Promise.all(
    options.dataTypes.map(type =>
      deleteDataByType(tenantId, type, options.olderThan)
    )
  );

  return {
    requestId: generateRequestId(),
    deletedAt: new Date(),
    recordsDeleted: results.reduce((a, b) => a + b, 0),
    dataTypes: options.dataTypes,
  };
}
```

Tenant Offboarding

```
async function offboardTenant(tenantId: string): Promise<void> {
  // 1. Export all data (required for compliance)
  const exportUrl = await exportTenantData(tenantId);

  // 2. Notify all users
  await notifyTenantUsers(tenantId, 'account_closing');

  // 3. Wait for grace period (30 days default)
  await scheduleTenantDeletion(tenantId, {
    gracePeriod: 30,
    exportUrl,
  });

  // 4. After grace period, delete all data
  // (Handled by scheduled job)
}
```

Legal Holds

Implementing a Legal Hold

```
-- Legal hold table
CREATE TABLE legal_holds (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tenant_id UUID REFERENCES tenants(id),
  user_id UUID REFERENCES users(id),
  hold_type VARCHAR(50) NOT NULL, -- 'litigation', 'investigation', 'regulatory'
```

```
  description TEXT,
  started_at TIMESTAMPTZ DEFAULT NOW(),
  expires_at TIMESTAMPTZ,
  created_by UUID REFERENCES administrators(id),
  CONSTRAINT legal_holds_target CHECK (tenant_id IS NOT NULL OR user_id IS NOT NULL)
);

-- Prevent deletion of held data
CREATE OR REPLACE FUNCTION check_legal_hold()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1 FROM legal_holds
    WHERE (tenant_id = OLD.tenant_id OR user_id = OLD.user_id)
    AND (expires_at IS NULL OR expires_at > NOW())
  ) THEN
    RAISE EXCEPTION 'Cannot delete data under legal hold';
  END IF;
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

Suspending Retention Policies

```
// Suspend automatic deletion during legal hold
async function applyLegalHold(params: {
  holdId: string;
  scope: 'tenant' | 'user';
  targetId: string;
}): Promise<void> {
  // Update retention flags
  await updateRetentionPolicy(params.targetId, {
    suspended: true,
    holdId: params.holdId,
  });

  // Exclude from cleanup jobs
  await excludeFromCleanup(params.targetId);

  // Notify compliance team
  await notifyCompliance('legal_hold_applied', params);
}
```

Audit Trail

Retention Actions Log

```
-- Log all retention-related actions
CREATE TABLE retention_actions (
```

```
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  action_type VARCHAR(50) NOT NULL, -- 'delete', 'archive', 'export', 'hold'
  target_type VARCHAR(50) NOT NULL, -- 'user', 'tenant', 'data_type'
  target_id VARCHAR(255) NOT NULL,
  records_affected INTEGER,
  performed_by UUID,
  reason TEXT,
  metadata JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Index for compliance queries
CREATE INDEX idx_retention_actions_date ON retention_actions(created_at);
CREATE INDEX idx_retention_actions_target ON retention_actions(target_type, target_id);
```

Compliance Reporting

```
// Generate retention compliance report
async function generateRetentionReport(
  startDate: Date,
  endDate: Date
): Promise<RetentionReport> {
  return {
    period: { start: startDate, end: endDate },

    // Data deleted by type
    deletions: await getRetentionActions('delete', startDate, endDate),

    // Data archived
    archives: await getRetentionActions('archive', startDate, endDate),

    // Active legal holds
    legalHolds: await getActiveLegalHolds(),

    // Policy violations (data past retention not deleted)
    violations: await getRetentionViolations(),

    // User deletion requests
    userRequests: await getUserDeletionRequests(startDate, endDate),
  };
}
```

Verification

Monthly Retention Audit

- ☐ Verify cleanup jobs running successfully
- ☐ Check for retention policy violations
- ☐ Review legal holds status

- ☐ Verify S3 lifecycle policies active
- ☐ Confirm CloudWatch log retention settings
- ☐ Review user deletion requests processed
- ☐ Update retention schedule if needed

Compliance Queries

```sql
-- Find data past retention period
SELECT
  'chat_messages' as table_name,
  COUNT(*) as records,
  MIN(created_at) as oldest_record
FROM chat_messages
WHERE created_at < NOW() - INTERVAL '90 days'
AND archived = false

UNION ALL

SELECT
  'api_keys' as table_name,
  COUNT(*) as records,
  MIN(revoked_at) as oldest_record
FROM api_keys
WHERE revoked_at < NOW() - INTERVAL '90 days';
```

Contact

Role	Contact	Purpose
Data Protection Officer	dpo@radiant.example.com	GDPR requests
Legal	legal@radiant.example.com	Legal holds
Compliance	compliance@radiant.example.com	Audit questions