AWS PostgreSQL Backup & Archival Architecture

1. Project Overview

This project is designed from a **Lead / Cloud Architect perspective**, prioritizing design decisions and recovery strategies over simple scripting.**Project Overview: Production-Grade PostgreSQL Backup and Archival on AWS RDS**

This project establishes a production-grade strategy for backing up and archiving PostgreSQL databases hosted on AWS RDS.

Key Goal: To design a **reliable, cost-effective, and highly recoverable backup system** that ensures business continuity and compliance for both historical and active PostgreSQL data.

Architecture Focus (Lead / Cloud Architect Perspective):

The design prioritizes strategic architecture decisions and comprehensive recovery plans over basic scripting, focusing on:

- **Long-Term Data Retention:** Enabling archival for data spanning 2023–2025 and beyond.
- **Disaster Recovery (DR):** Ensuring robust preparedness for data loss scenarios.
- **Operational Excellence:** Implementing automation and clearly defining operational ownership.
- **Visibility:** Providing clear documentation and a transparent view of the backup architecture.

2. Architecture

The system leverages **Amazon RDS** for hosting the PostgreSQL database. Data durability and recovery are ensured through a robust backup strategy:

- **Logical Backups** are performed using the standard `pg_dump` utility.
- Backups are securely stored in **Amazon S3**.
- **S3 Glacier** is utilized for long-term archiving of older backups, managed via lifecycle policies.
- **GitHub Actions** automates the entire backup process.
- Comprehensive documentation outlines all monitoring and recovery procedures.

3. AWS Setup Steps (Console-Based)

Steps to Create the Backup S3 Bucket:

1. Navigate to the **AWS Console** and select **S3**.
2. Click the **Create bucket** button.
3. Configure the following settings:
    - **Bucket name:** `pg-backup-archival-bucket`
    - **Region:** Select the same region as your RDS instance.
4. Ensure **Block public access** is enabled.
5. Enable the following features:
    - Versioning
    - Default encryption (using either SSE-S3 or SSE-KMS)
6. Complete the process by clicking **Create bucket**.

This reduces storage costs while keeping data retrievable.

3.2 Optimize Costs with S3 Lifecycle Policies

To reduce storage expenses while maintaining data accessibility, configure lifecycle policies on your S3 bucket:

1. Navigate to the S3 bucket's **Management** section.
2. Select **Lifecycle rules** and create a new rule.
3. Set the policy to:
    - Transition objects to **Glacier after 30 days**.
    - (Optional, for compliance) Configure object deletion **after 2 years**.

To ensure proper configuration of the RDS PostgreSQL instance, the following requirements must be met:

- **Backup Strategy:**
    - Automated backups must be enabled.
    - The backup retention period must be a minimum of 7 days.
    - Snapshots should be enabled specifically for major software releases.
- **Networking:** The RDS instance must be deployed within a **private subnet**.

3.3 IAM & Security Best Practices

- **Principle of Least Privilege:**
    - Grant only the necessary permissions (e.g., S3 write access, CloudWatch logging).

- **Access Management:**
    - Utilize IAM roles for AWS access instead of storing access keys.
- **Secret Management:**
    - Do not store sensitive secrets directly in GitHub.

4. Backup Strategy

This project employs a **multi-layered backup approach** to ensure resilience against various failure scenarios.4.1 Backup Methods

Backup Type	Primary Function
RDS Automated Backups	Facilitates point-in-time recovery.
Manual Snapshots	Provides a safety net before major deployments.
Logical Backups (`pg_dump`)	Allows for granular restoration of specific data elements.
S3 Archival	Used for long-term data retention.

4.2 Logical Backup Process

1. A backup job is initiated by **GitHub Actions**.
2. The `pg_dump` utility exports the database content.
3. The resulting backup file is uploaded to **S3**.
4. **Lifecycle policies** manage the archiving of this data to Glacier.

This logical backup method is critical as it enables the restoration of **individual tables or schemas**, a capability not provided by standard snapshots.

5. Automation Using GitHub Actions

GitHub Actions is the platform chosen to simulate scheduled backups, offering several advantages for this automation.

Benefits of Using GitHub Actions:

- **Cost Efficiency:** No incurring additional AWS costs.
- **CI/CD Integration:** Leverages a CI/CD-native environment.

- **Simple Scheduling:** Uses cron expressions for easy scheduling.
- **Demonstrates Proactive Automation:** Highlights an automation-focused development approach.

Key Automation Features:

- **Scheduled Execution:** Runs backups automatically via cron.
- **On-Demand Testing:** Allows for manual triggers to facilitate testing.
- **Secure Simulation:** Ensures safe execution without exposing real secrets.

6. Disaster Recovery Strategy

6.1 Recovery Objectives

Metric	Target
Recovery Point Objective (RPO)	24 hours
Recovery Time Objective (RTO)	4 hours

6.2 Recovery Scenarios

Scenario	Recovery Action
Accidental Data Deletion	Restore from `pg_dump` backup stored in S3.
RDS Instance Failure	Restore from the latest RDS snapshot and attach to a new RDS instance.
AWS Region Failure	Utilize cross-region S3 replication (Planned for future enhancement).

7. Operations & Monitoring

7.1 Backup Validation Process

- Execute a full restore test monthly.
- Validate the integrity and consistency of the restored database.

- Document and report the success or failure of the restore test.

7.2 Monitoring Strategy

- Track S3 storage consumption and performance metrics.
- Monitor execution logs for backup jobs.
- Implement alerting via CloudWatch (Planned for future enhancement).

8. Security & Compliance

- **Credential Management:** No storage of sensitive credentials within the codebase.
- **Data Protection:** Enforce encryption for data both at rest and in transit.
- **Access Control:** Implement IAM policies based on the principle of least privilege.
- **Network Isolation:** Utilize private networking for the RDS database instance.

9. Design Decisions & Trade-Offs

Decision	Rationale
Logical Backups (`pg_dump`)	Provides flexibility for granular object-level recovery.
S3 Standard & Glacier	Optimizes storage costs based on data access frequency.
GitHub Actions	Enables lightweight, integrated, and easily maintained automation.
Documentation-First Approach	Ensures maximum operational clarity and ease of maintenance.

10. Scope & Assumptions

10.1 In Scope

- Definition of the Backup Architecture.
- Development of the Recovery Strategy.
- Design for cost-aware and optimized storage.
- Design and implementation of backup automation.

10.2 Out of Scope

- Use of real production access credentials.
- Execution of a full, production-scale restore.
- Multi-region Disaster Recovery (Planned for future enhancement).

11. Future Enhancements

- Integration with native AWS Backup services.
- Implementation of cross-region data replication.
- Transition to Lambda-based serverless automation.
- Set up notification alerting via AWS SNS.

12. Supporting Resources

- **GitHub Repository:** 🔗 https://github.com/DevOpsJen/aws-postgresql-backup-archival

13. Final Perspective

This document aims to illustrate **how a Lead or Architect approaches** a critical service like database backups, encompassing:

- **Restorability:** Moving beyond merely *taking* backups to proving they can be *restored*.
- **Cost Management:** Analyzing and optimizing the total cost of ownership.
- **Operational Reality:** Defining how the system is monitored, maintained, and operated in a production environment.