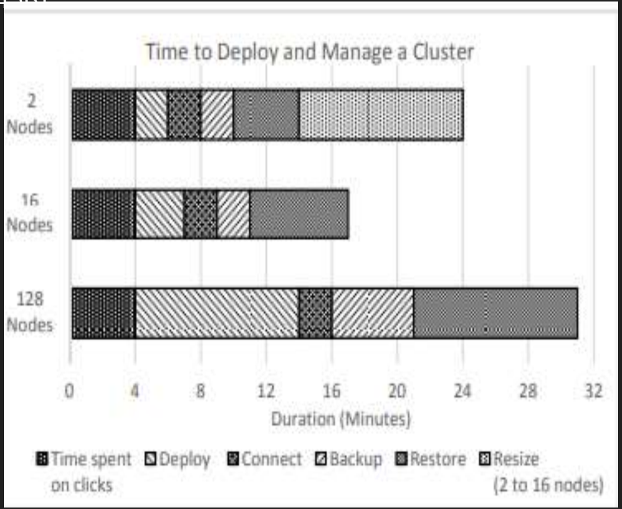
Amazon Redshift
&
The Case for Simpler Data Warehouses

ABSTRACT

- ➔ Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse solution
- ➔ In this, we discuss an oft-overlooked differentiating characteristic of Amazon Redshift – simplicity. Our goal with Amazon Redshift was not to compete with other data warehousing engines, but to compete with non-consumption.
- ➔ Most database vendors target larger enterprises, there is little correlation in today's economy between data set size and company size.
- ➔ Amazon Redshift was designed to bring data warehousing to a mass market by making it easy to buy, easy to tune and easy to manage while also being fast and cost-effective.

Introduction to Amazon Redshift

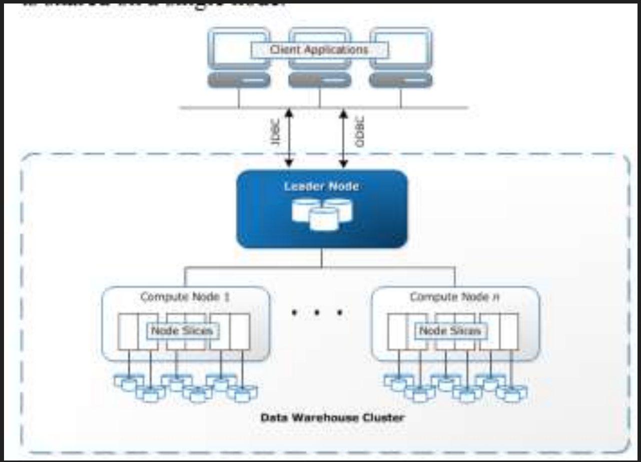
- Data Warehouse Market Overview:
  - Data warehouse market estimated at 1/3 of the relational database market (\$14B vs. \$45B).
  - Compound Annual Growth Rate (CAGR) of 8-11%.
- Data Explosion Challenge:
  - Data storage at enterprises growing at 50-60%, doubling every 20 months.
  - Most data is "dark data" - collected but not easily analyzed.
- Analysis Gap Challenges:
  - Cost, complexity, performance, and rigidity hinder timely data analysis.
  - Existing solutions struggle with growing datasets.
- Amazon Redshift Solution:
  - Augments transaction-processing databases with petabyte-scale clusters.
  - Overcomes challenges, demonstrating efficiency with large-scale examples.
- Key Success Factors:
  - Utilizes familiar data warehousing techniques (columnar layout, compression, etc.).
  - Additional design goals: cost-effective experimentation, rapid deployment, minimized administration, scaling concerns, and tuning.
- Operational Benefits:
  - Leverages Amazon EC2, S3, and VPC for pricing, durability, and security.
  - Automatic deployment and patching enable rapid feature addition and agile operation.
- Paper Focus:
  - Overview of Amazon Redshift's system architecture, simplicity, and lessons learned in two years of operation.



Amazon Redshift System Architecture

- Architecture Overview:
  - Data Plane: Consists of the database engine for SQL execution, supporting analytics workloads.
  - Control Plane: Manages workflows, monitors, and maintains the database.
  - Relies on other AWS services for data and control plane execution.
- Data Plane - Amazon Redshift Engine:
  - SQL-compliant, massively parallel, designed for analytics workloads.
  - Leader node and compute nodes for scalability; supports single-node design.
  - Data distributed across nodes; compute nodes handle heavy lifting.
- Data Storage and Compute:
  - Compute nodes partitioned into slices for parallel processing.
  - Data storage is distributed across compute nodes for scale-out.
  - Data blocks replicated within the database instance and in Amazon S3 for durability.
- Query Processing:
  - Leader node generates and compiles query plans.
  - Executable and plan parameters sent to compute nodes.
  - Compute nodes perform operations in parallel; results aggregated at the leader node.
- Data Loading:
  - Special case of query processing using a modified COPY command.
  - Parallelized across slices; supports loading from Amazon S3, DynamoDB, EMR, or SSH.
  - Supports compression schemes, optimizer statistics, JSON data, encryption, and compression.

- Control Plane:
  - Each node has host manager software for deploying, monitoring, and managing databases.
  - Host manager software assists in aggregating events, generating instance-level events, and archiving logs.
  - Most control plane actions coordinated by a separate fleet for fleet-wide monitoring and maintenance tasks.
- Dependent AWS Services:
  - Leverages various AWS services, including EC2 for instances, S3 for backup, SWF for control plane actions, CloudWatch for metrics, SNS for alarms, VPC for network isolation, Route53 for DNS, CloudTrail for audit logging, and Key Management Service and CloudHSM for key management.
  - Internal AWS services used for deployment, credentials, log harvesting, load balancing, and more.
- Integration for Accelerated Development:
  - Integration of traditional parallel, distributed relational database architecture with service-oriented architecture.
  - Leverages AWS services for intrusion detection, network QoS, server health monitoring, and more.
- Backup and Restore Differentiation:
  - Automation of continuous, incremental, and automatic backup using Amazon S3.
  - Streaming restore capability for SQL operations during background block retrieval.
  - Enables performant queries in a fraction of the time compared to full restore.
- Customer Practices:
  - Significant number of customers delete clusters every Friday and restore from backup each Monday, leveraging the efficiency of Amazon Redshift's backup and restore capabilities.



The Case for Simplicity

- Simplicity in Success:
  - SQL-based databases succeeded due to declarative query processing and concurrent execution.
  - Data warehousing and columnar layouts extended simplification in schema design and join processing.
- Balancing Power and Understanding:
  - Additional power requires additional education and understanding.
  - Strive for a balance – empower without overwhelming users with complexity.
- Simplifying Purchase Decision Process:
  - "Time to first report" crucial metric; starts when customer evaluates the service.
  - Linear pricing model, standard drivers, and reduced upfront steps simplify cluster creation.
  - Continuous improvement to reduce creation time (e.g., preconfigured nodes) and enhance customer experience.
- Reducing Cost of Error:
  - Customers can experiment easily with free trial and hourly pricing for reduced commitment.
  - Resize clusters up or down at any time; no need for upfront capacity and performance estimation.

- Simplifying Database Administration:
  - Majority of Amazon Redshift customers lack a designated DBA.
  - Service manages undifferentiated heavy lifting in administration, including provisioning, patching, monitoring, repair, backup, and restore.
  - Declarative administration operations for parallelization and distribution.
- Efficient Backup and Restore:
  - Automatic system backups with automatic aging; user backups leverage existing system backups.
  - Disaster recovery through checkbox selection for a second region.
  - Streaming restore capabilities enable quick query issuance after initiating a cluster in the remote region.
- Encryption Simplicity:
  - Enabling encryption as simple as setting a checkbox and specifying a key provider.
  - Multi-layered encryption ensures security, with easy key rotation and repudiation.
  - Leverages Amazon VPC for network isolation, enhancing security features.
- Future Directions:
  - Aim to eliminate user-initiated table administration operations, making them akin to backup in operation.
  - Database should autonomously detect data access performance degradation and take corrective action during light load.
- Simplifying Database Tuning:
  - Amazon Redshift has few tuning knobs, placing responsibility on the service.
  - Instance type, number of nodes, sort, and distribution model set by the customer; other parameters set accurately by the service.
  - Ongoing efforts to reduce the need for customer-initiated settings like sort column and distribution key.
  - Emphasis on making tuning knobs "dusty with disuse" to minimize customer burden.
- Towards Elasticity:
  - Exploration of a more elastic system, allowing growth and shrinkage as load requires.
  - Continuous focus on simplicity, aiming to balance power and user understanding.

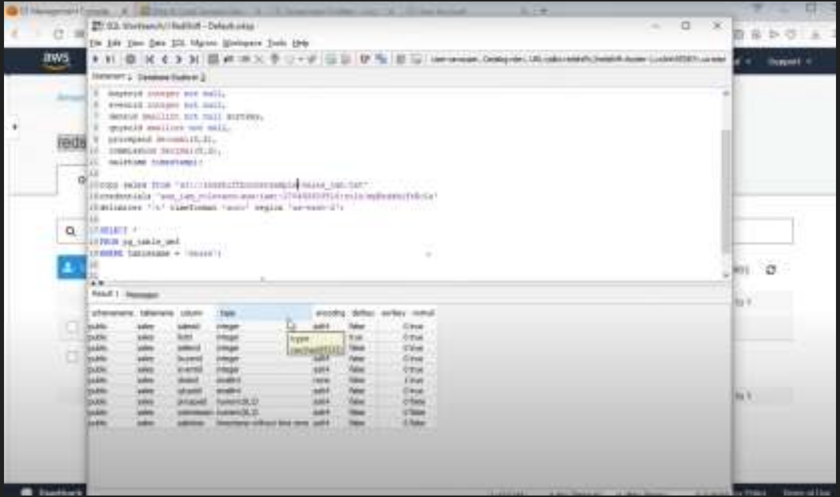
Customer Use Cases

- **Enterprise Data Warehousing:**
    - Traditional use case: hourly or nightly data ingestion from source relational databases.
    - Access through BI tools.
    - Appreciate simplicity and transparency in procurement.
    - Struggling with maintenance overhead in existing systems.
    - Value managed systems handling undifferentiated heavy lifting.

- . Semi-structured "Big Data" Analysis:
    - Integrated analysis of log and transaction data.
    - Migration from HIVE on Hadoop for better performance and lower cost.
    - Direct availability to business analysts using SQL or BI tools.
    - Simplicity crucial as they lack dedicated DBAs.
    - Opportunities to simplify further, e.g., support transient data warehouses.

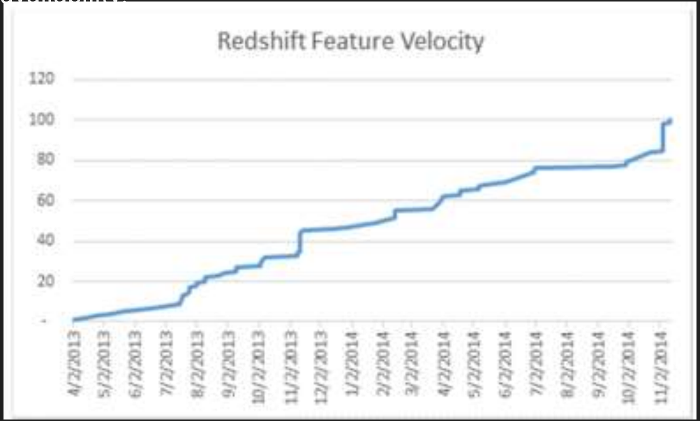
- Data Transformation:
    - Integration into data processing pipelines.
    - Usage in ad-tech for converting billions of ad impressions into lookup tables.
    - Direct integration into customer-facing screens under analytic reports and graphs.
    - Move towards SQL for straightforward and declarative intent indication.

- **Small Data:**
    - Customers without prior data warehousing experience.
    - Adoption of Amazon Redshift for improved performance, OLTP system offload, and history retention.
    - Short time lag to source data changes.
    - Importance of automated change data capture and automatic schema creation and maintenance.

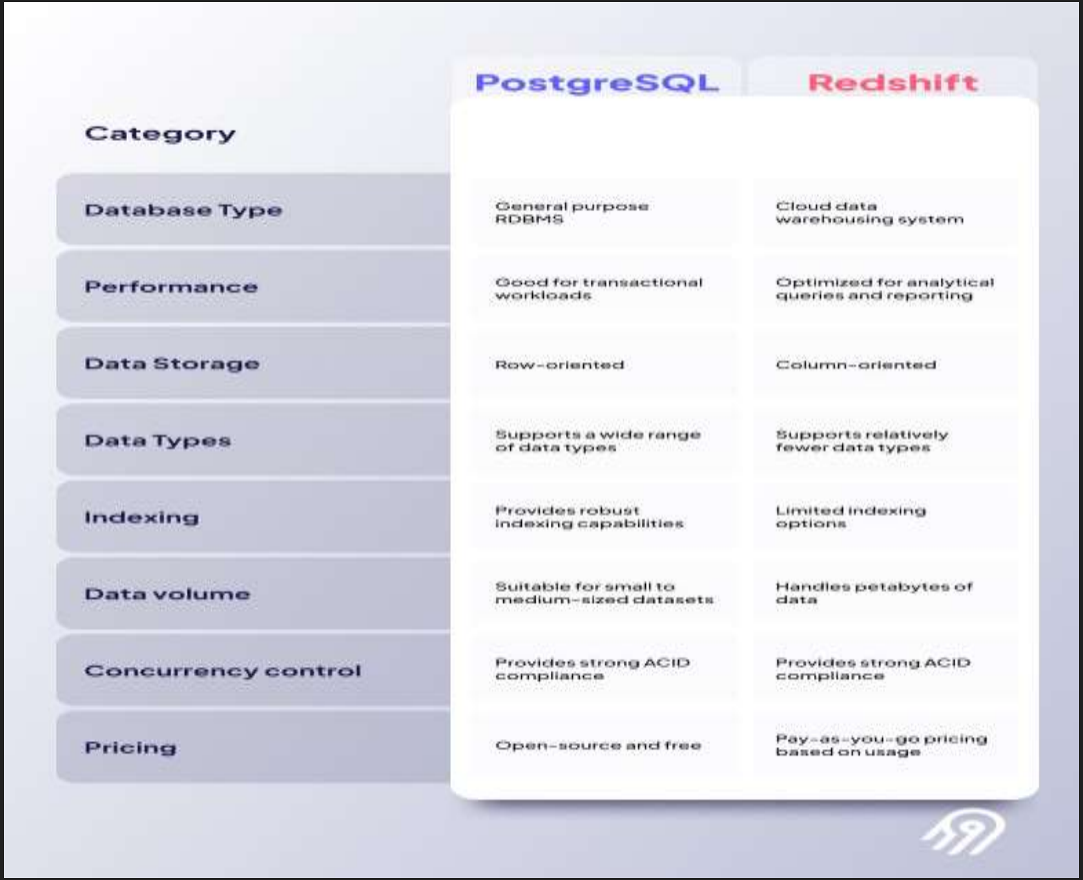


Lessons Learned

- Design for Degrading Failures:
  - Prioritize designs that degrade gracefully on failures rather than outright loss of availability.
  - Replicate data blocks to mask issues with disks.
  - Extend mitigation strategies for software or service dependencies.
- Continuous Delivery to Customers:
  - Adopt continuous delivery not just to staging but directly to customers.
  - Weekly automatic patching of customer clusters with reversible patches.
  - Frequent, small patches improve reproducibility and issue diagnosis.
- Use Pareto Analysis for Work Scheduling:
  - Operational load correlates with business success.
  - Page on each database failure to address customer concerns.
  - Conduct Pareto analysis on error logs to address top ten causes weekly.
- Understand Customer Functional Requirements:
  - Conduct over 1000 direct one-to-one conversations with customers annually.
  - Obtain actionable telemetry for scheduling feature development.
  - Aim to automate usage statistics collection for more insights into customer needs and service gaps.



PostgreSQL vs Redshift



Category	PostgreSQL	Redshift
Database Type	General purpose RDBMS	Cloud data warehousing system
Performance	Good for transactional workloads	Optimized for analytical queries and reporting
Data Storage	Row-oriented	Column-oriented
Data Types	Supports a wide range of data types	Supports relatively fewer data types
Indexing	Provides robust indexing capabilities	Limited indexing options
Data volume	Suitable for small to medium-sized datasets	Handles petabytes of data
Concurrency control	Provides strong ACID compliance	Provides strong ACID compliance
Pricing	Open-source and free	Pay-as-you-go pricing based on usage

CONCLUSION

- Price, Performance, and Simplicity:
  - Amazon Redshift excels in price, performance, and simplicity.
  - Extends data warehousing use cases to big data and SaaS applications with embedded analytics.
- Instant Provisioning and Scalability:
  - Clusters provisioned in minutes, enabling easy scalability.
  - No upfront payments, allowing customers to start with no commitments and scale up to petabyte clusters.
- Automated Operations and Security:
  - Automated patching, provisioning, scaling, securing, backup, restore.
  - Comprehensive security features: encryption at rest, in transit, HSM integration, and audit logging.
- Changing the Landscape:
  - Redefining data warehousing for traditional enterprises and new segments.
  - Empowering enterprises like NTT DOCOMO and Amazon.com to startups like Pinterest and Flipboard.
- Accessibility and Adoption:
  - Lowering the cost and effort associated with deploying data warehousing systems.
  - Making data warehousing technology accessible to a broader audience.