Achieving High Availability (HA) in PostgreSQL
Strategies, Tools, and Best Practices

Principal Engineer

Ibrar Ahmed

Postgres
Conference 2024
San Jose / United States
April 17, 2024

pgEdge

High Availability

Operational Continuity: High availability in PostgreSQL guarantees that the database remains operational and accessible, minimizing downtime even during system failures.

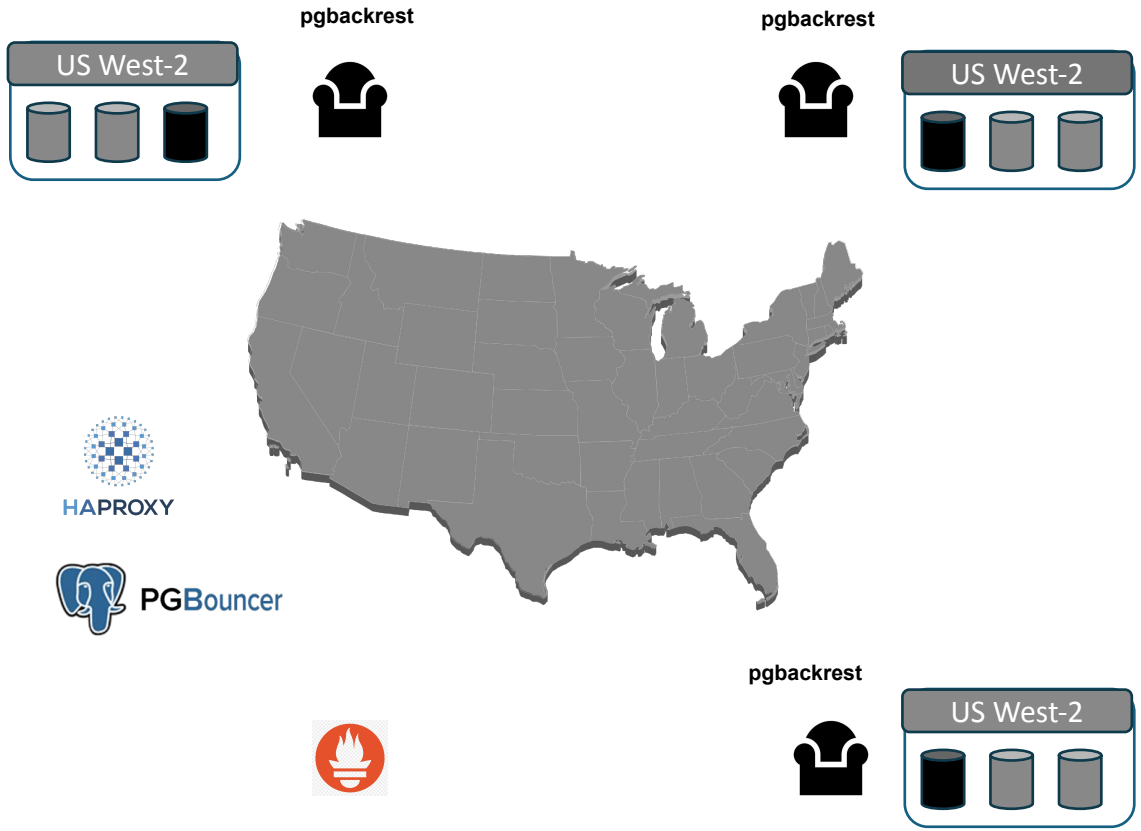
Redundancy and Automated Failover: Implement redundant system configurations alongside automated failover processes to ensure uninterrupted service and quick recovery from failures.

Data Integrity with Synchronous Replication: Utilize synchronous replication to maintain data integrity, ensuring that all data on the primary server is exactly mirrored on the standby server at all times, thus preventing data loss.

Seamless Failover Mechanisms: Develop and maintain seamless failover mechanisms that enable quick and efficient transition to standby servers without service disruption, ensuring continuous database availability.

Monitoring and Regular Testing: Regularly monitor the health of the database systems and conduct failover drills to ensure the high availability setup performs effectively when actual failures occur.

High Availability in PostgreSQL



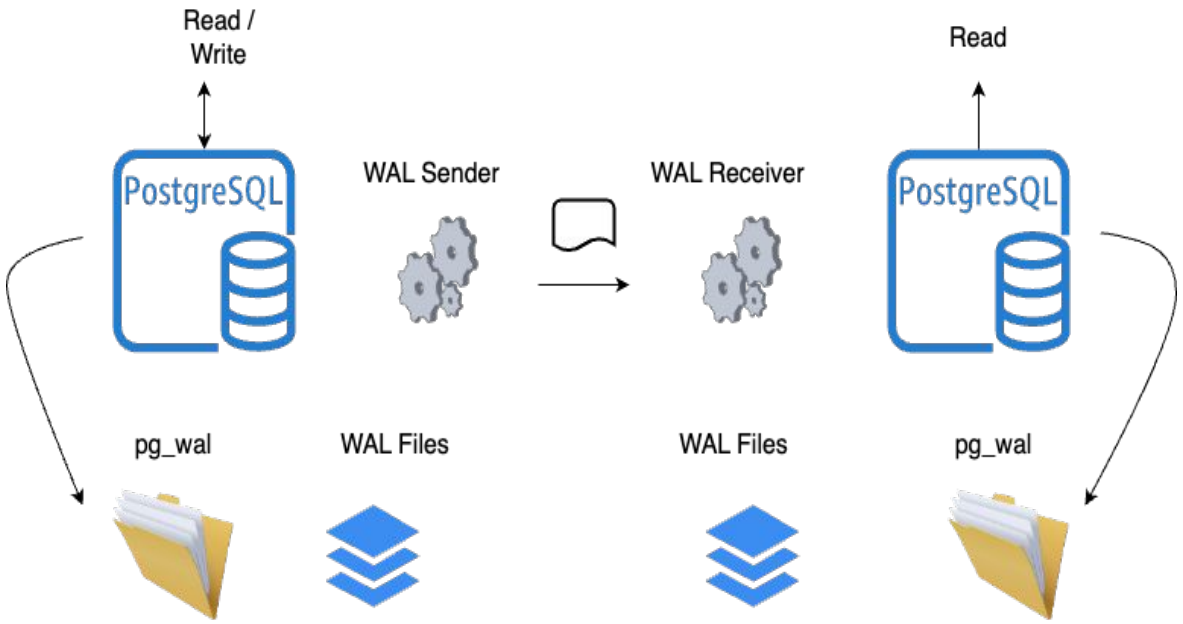
- High availability in PostgreSQL ensures the database is operational and accessible without significant downtime.
- **Replication**: Streaming replication is used to create standby servers that are continuously updated with data from the primary server.
- **Failover**: Automatic failover process to switch to a standby server in case the primary server fails.
- **Load Balancing**: Distribution of queries across multiple servers to improve performance and distribute the workload.
- **Connection Pooling**: Management of database connections to optimize resource usage and improve performance.
- **Monitoring and Management**: Continuous monitoring of database servers to detect and respond to issues promptly, often using tools like pgBouncer and pgpool.
- **Backup and Recovery**: Regular backups and robust recovery plans to protect against data loss and ensure quick service restoration.
- **Clustering**: Grouping multiple servers to work as a single system, providing redundancy and improving availability.

Replication

PostgreSQL supports several replication methods, including logical and streaming each catering to different requirements and use cases.

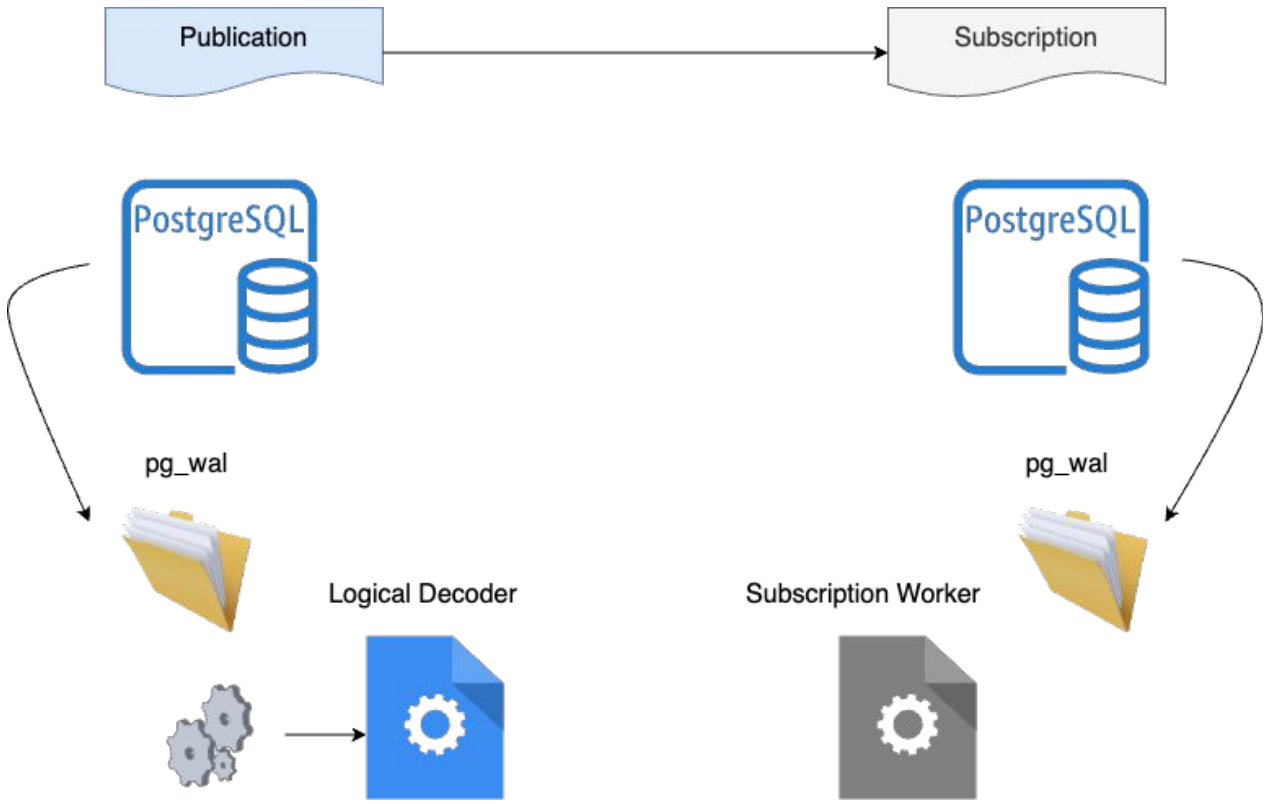
- **Streaming Replication:** A popular method for real-time replication, streaming replication involves a primary server sending data changes to one or more standby servers. This method is helpful for high availability and load balancing.
- **Synchronous vs. Asynchronous Replication:** In synchronous replication, transactions must be confirmed by both the primary and standby servers before being committed, ensuring data consistency but potentially affecting performance. Asynchronous replication, while faster, does not guarantee immediate consistency across servers.
- **Logical Replication:** Allows selective data replication at the table level, allowing the flexibility to replicate only specific tables or rows. It's beneficial for upgrading systems with minimal downtime and integrating data across different PostgreSQL versions.

Physical Replication



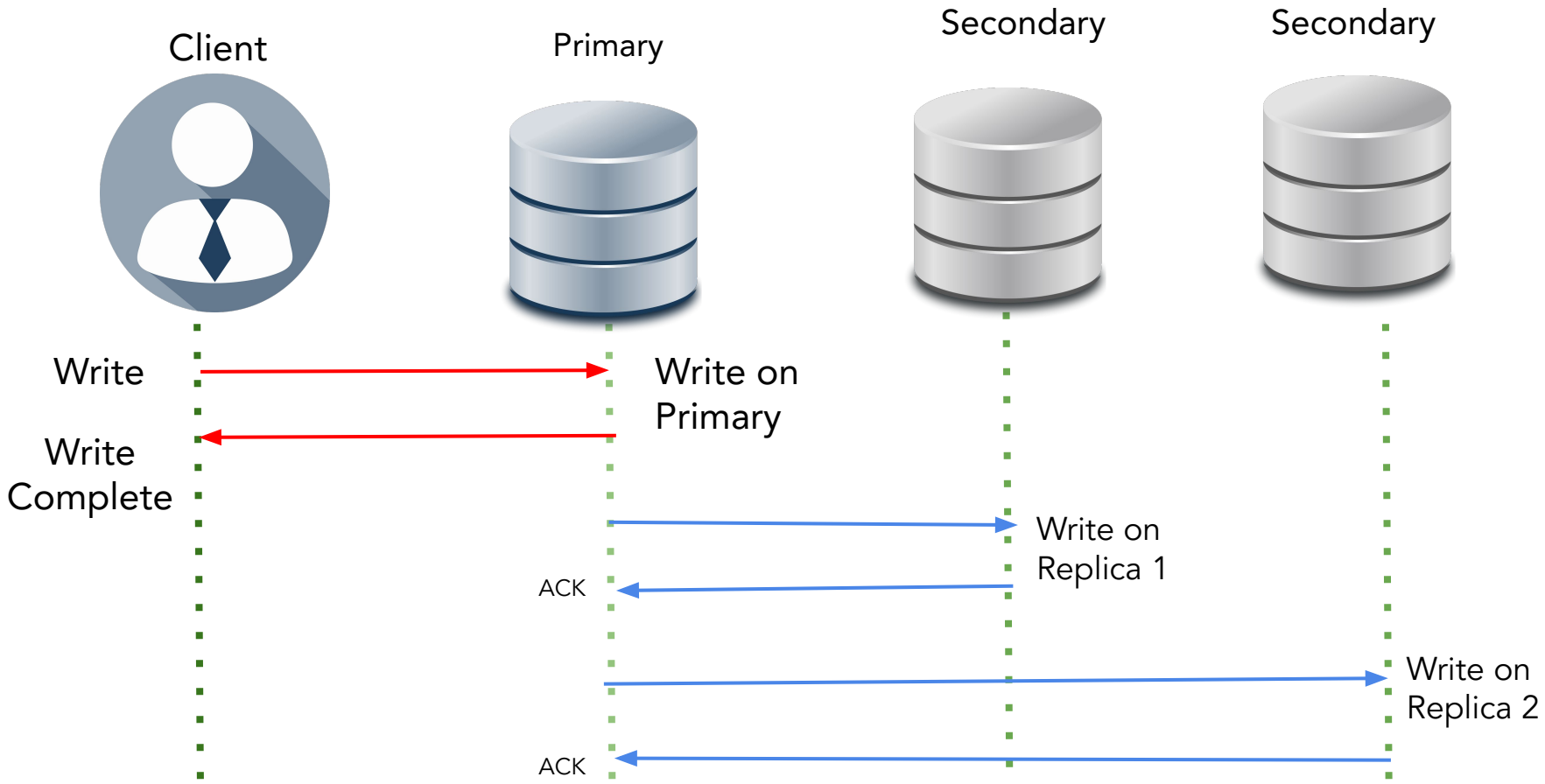
- Physical replication in PostgreSQL is a method for copying and synchronizing data from a primary server to standby servers in real-time.
- Real-time transfer of WAL records from a primary to standby servers to ensure data consistency and up-to-date replicas.
- Standby servers can run in hot standby mode, allowing them to handle read-only queries while replicating changes.
- Configurable as either synchronous, for strict data integrity, or asynchronous, for improved write performance.
- Facilitates automatic failover by promoting a standby to primary in case of primary server failure.

Logical Replication



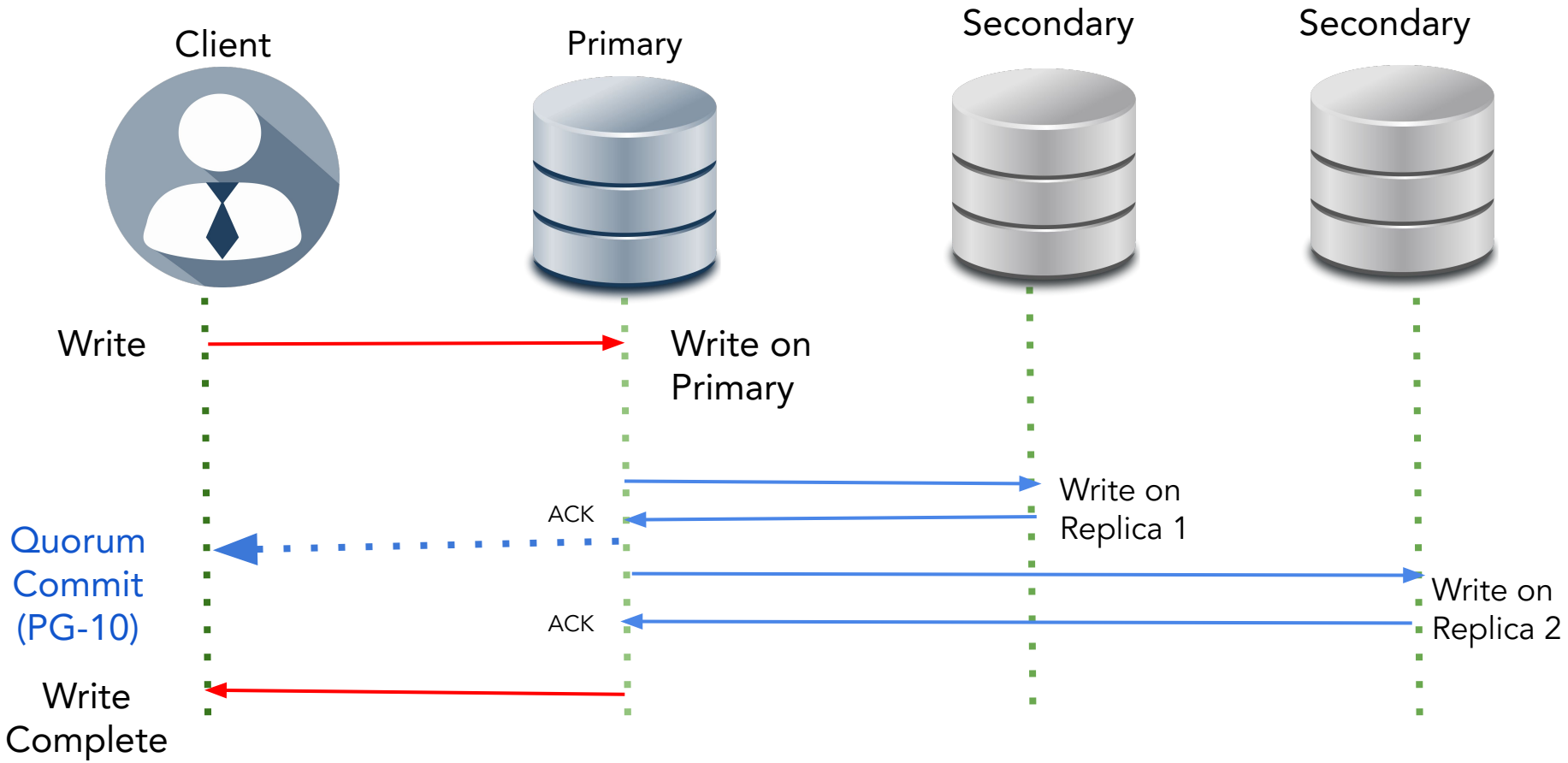
- Logical replication is method of copying data objects and changes based on replication identity.
- Provides fine grained control over data replication and security.
- Publisher / Subscriber model - one or more subscriber subscribe to one or more publisher.
- Copy data in format that can be interpreted by other systems using logical decoding plugins.
- Publication is set of changes generated from a table or group of tables.
- Subscription is the downstream end of logical replication.

Asynchronous Replication



pgEdge

Synchronous Replication



pgEdge

Deployment Models (Active-Standby / Active-Active)

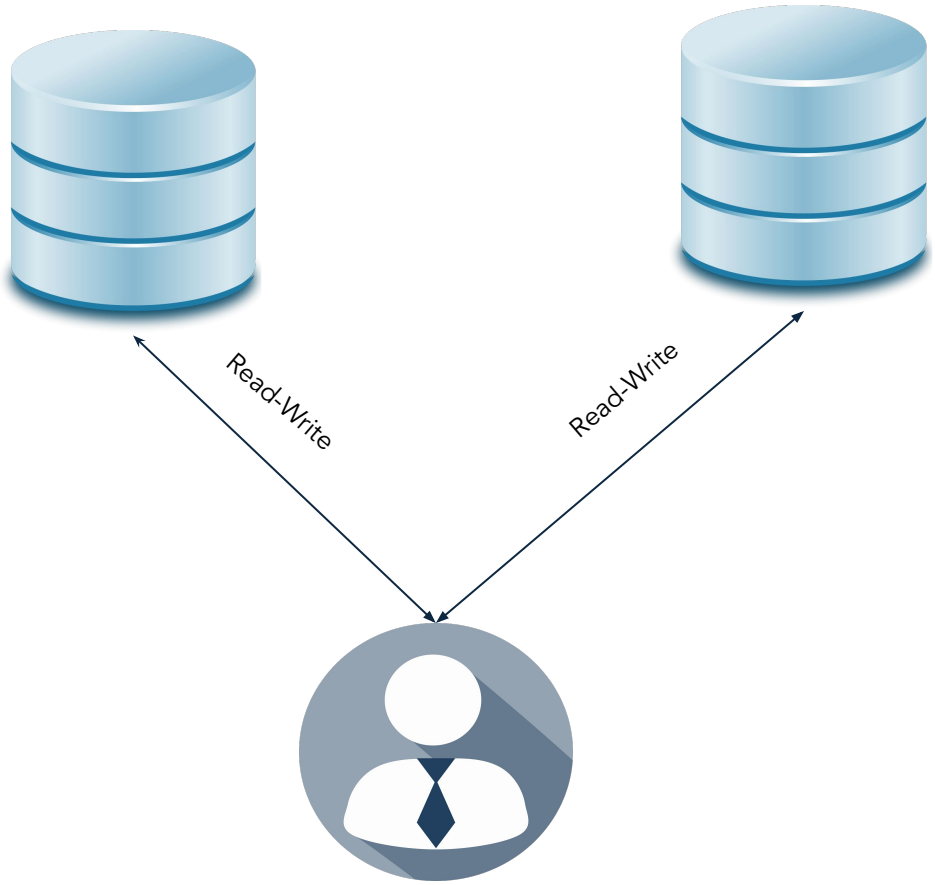
Active - Standby

- One primary and one or more stand-by servers
- Write traffic on primary and read traffic load balance on read replica using external tools
- Synchronous / Asynchronous / Quorum Commit choices
- Load Balancing / High Availability
- Automatic Failover using external tools
- Low chances of data loss

Active-Active

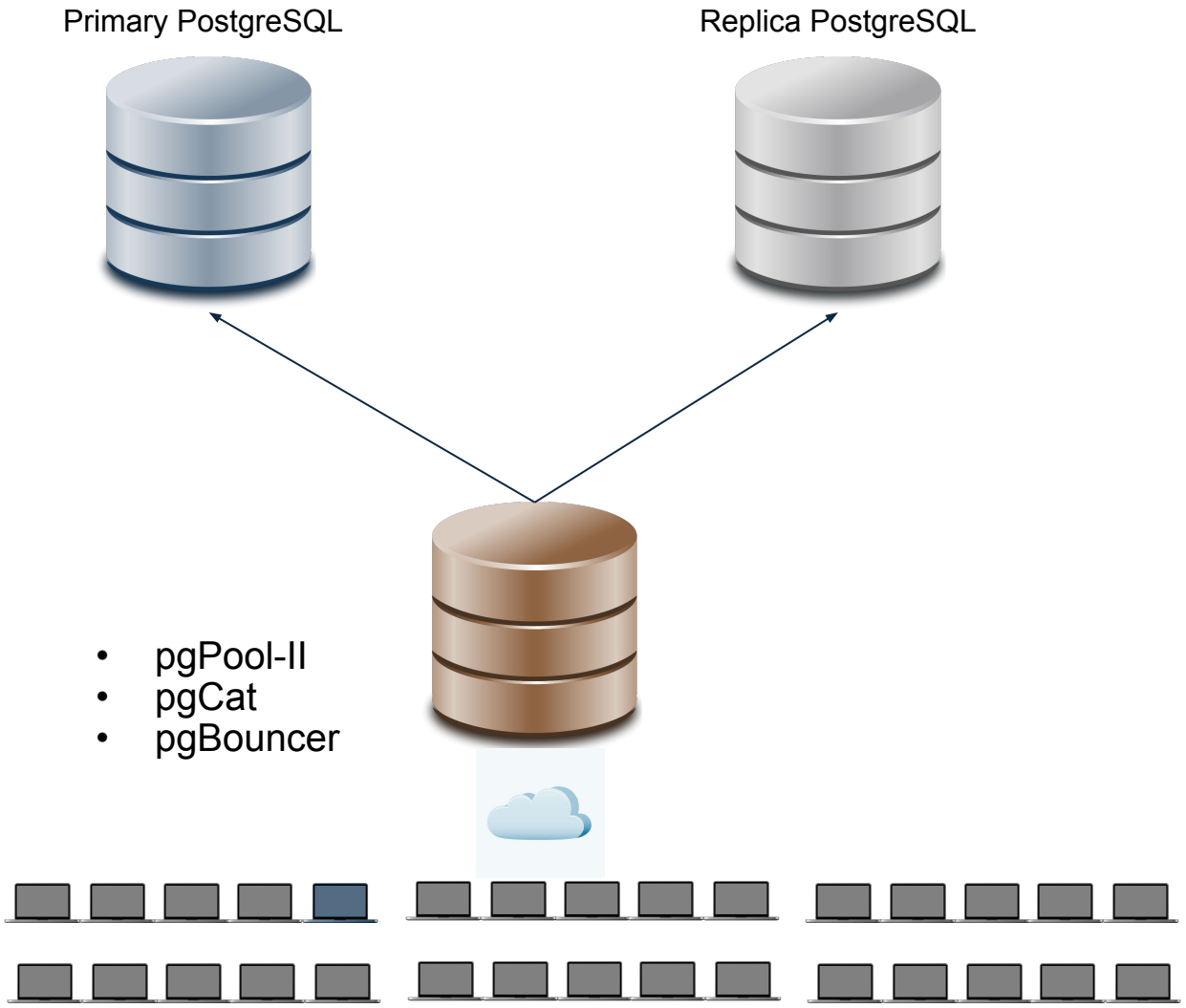
- One or more primary/active servers replicating between each other.
- Not part of Core PostgreSQL, implemented using 3rd party tools and extension
- Requires Conflict detection and resolutions
- Use cases are High Availability, data residency, data latency and near zero downtime upgrades

Multimaster Replication



- **Simultaneous Data Writing:** Multi-master replication allows multiple database instances to handle write operations simultaneously, enhancing the database's write availability and scalability.
- **Conflict Resolution**: Incorporates mechanisms to handle conflicts that arise when the same data is modified at different nodes, ensuring data consistency across all nodes.
- **Load Distribution**: Distributes both read and write loads across several nodes, effectively balancing the load and improving overall system performance.
- **Improved Fault Tolerance**: Increases the database system's fault tolerance by allowing the system to remain operational even if one of the master nodes fails, thereby reducing potential downtime.
- **Real-Time Data Synchronization**: Ensures real-time or near-real-time synchronization between nodes, keeping the databases up-to-date and consistent with each other.
- **Geographical Distribution**: Supports geographical distribution of database nodes, which can reduce latency for globally distributed applications by allowing users to interact with the nearest database node.

Connection Pooling



Reduced Connection Overhead: Connection pooling in PostgreSQL minimizes the time and resources required to establish database connections by reusing existing connections.

Enhanced Scalability: By managing a pool of active connections, PostgreSQL can handle more simultaneous client requests, improving scalability.

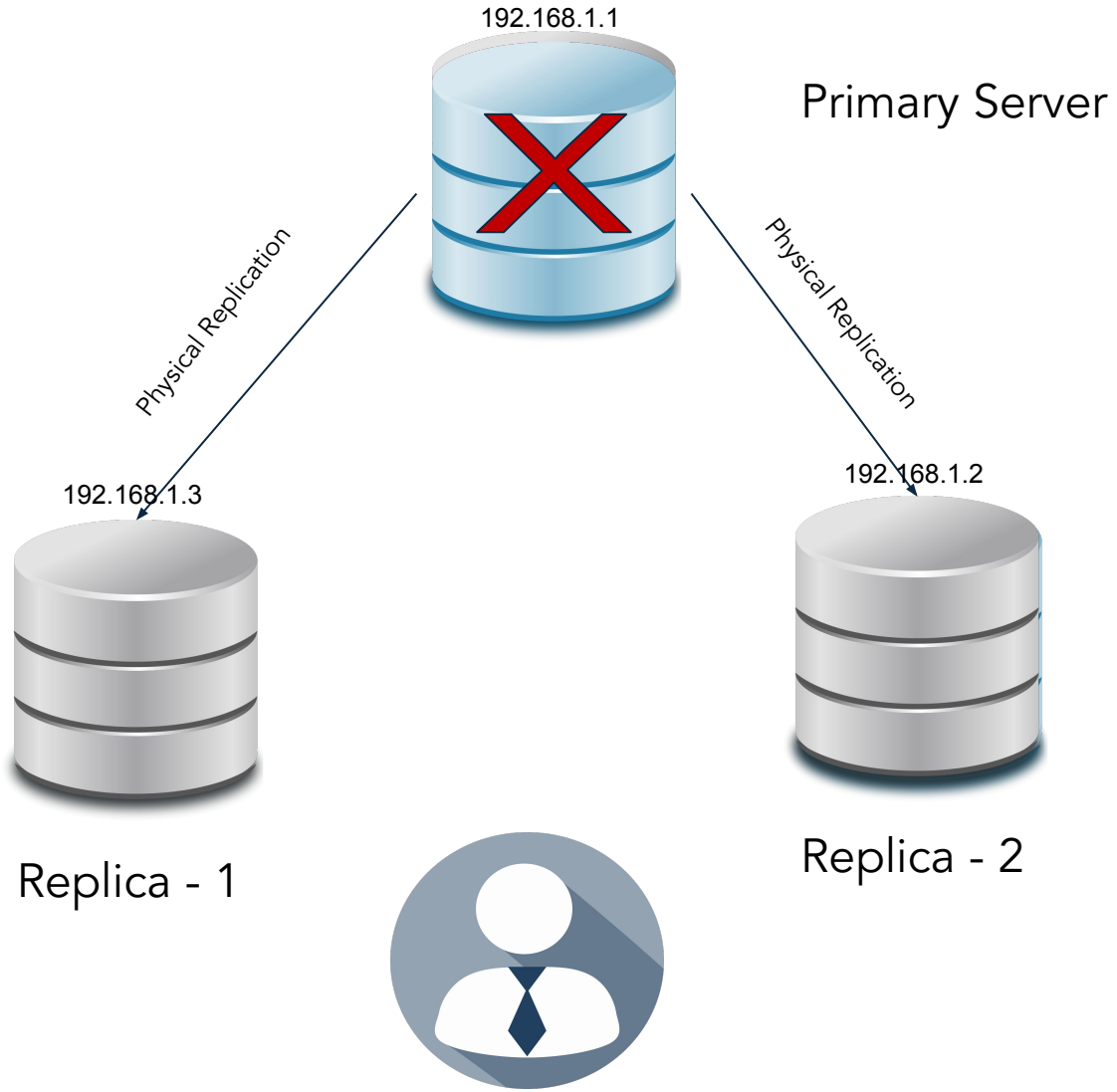
Improved Performance: Connection pooling leads to faster database access and response times, enhancing overall application performance.

Configurable Pooling Solutions: PostgreSQL supports external connection pooling solutions like PgBouncer,pgCat and Pgpool-II, which offer configurable settings for optimized performance.

Tools for High Availability in PostgreSQL

- **Patroni**: Automates PostgreSQL cluster management, handling failover and ensuring seamless transitions during node failures to maintain high availability.
- **ETCD:** A key component of Patroni, ETCD serves as a highly reliable distributed key-value store that manages the state of the cluster, facilitating consensus and leader election.
- **pgBouncer:** A lightweight connection pooler for PostgreSQL that reduces connection overhead and improves resource utilization by managing client connections.
- **Pgpool-II**: Enhances PostgreSQL performance by providing connection pooling, load balancing, and replication services, optimizing read operations and system resilience in high traffic environments.

Failover



Promote the Standby to Primary

- Use the `pg_ctl promote` command or create a trigger file if configured to use one.

```
# pg_ctl promote -D /usr/local/pgsql/data
```

- This action will convert the standby into a primary, and it will begin accepting write operations.

Reconfigure Application Connections

- Redirect all application connections from the old primary to the new primary (the promoted standby).
- Ensure client connection strings or load balancers are updated to point to the new primary server

Ensuring Minimal Service Disruption

- **High Availability Configuration:** Set up PostgreSQL with high availability clusters using tools like Patroni, which automate failover and recovery processes to ensure minimal service disruption.
- **Regular Backup and Restore Tests:** Implement routine backups using tools like pgBackRest or Barman, and regularly test restore processes to ensure data can be recovered quickly and accurately after an outage.
- **Connection Pooling:** Utilize connection pooling mechanisms such as pgBouncer or Pgpool-II to manage client connections efficiently, ensuring the database can handle high loads and maintain performance during peak times.
- **Load Balancing:** Deploy load balancers like HAProxy or utilize Pgpool-II to distribute database requests evenly across servers, reducing the risk of overloading any single server and maintaining service availability.
- **Real-Time Monitoring and Alerts:** Implement comprehensive monitoring with tools like Prometheus and Grafana to track database performance and system health, enabling quick response to issues before they cause significant service disruptions.

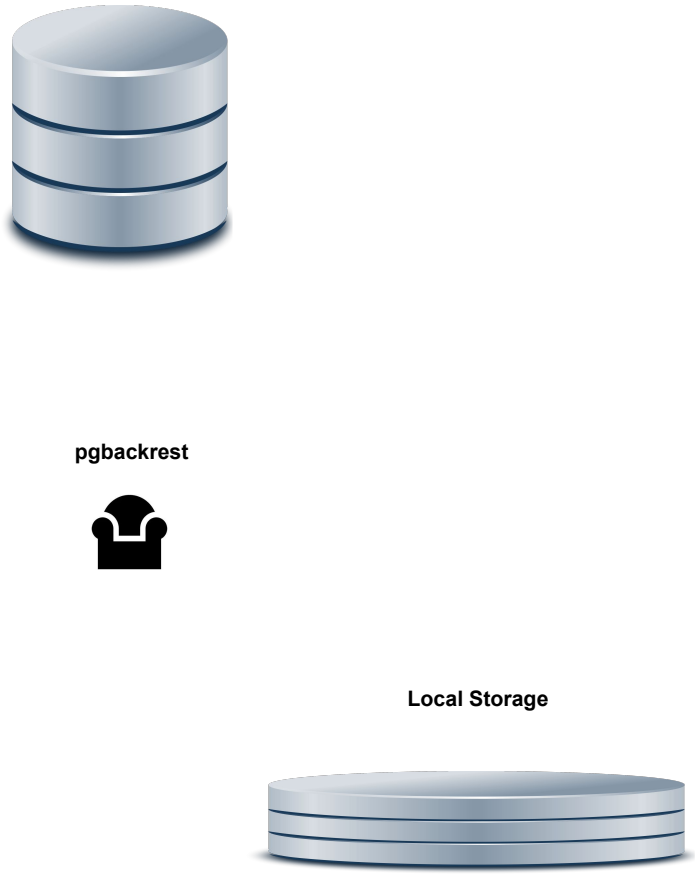
Monitoring

- **Comprehensive Logging:** PostgreSQL can be monitored through its detailed logging system, enhanced by tools like pgBadger, which analyzes logs and generates performance reports.
- **Built-in Statistics Collector**: Use PostgreSQL's built-in statistics collector for insights, with tools like pg_stat_statements and pg_stat_activity to analyze query performance and session activity.
- **Performance Dashboard Tools**: pgAdmin provides a comprehensive graphical interface for PostgreSQL management and monitoring, while PHPPgAdmin offers additional web-based monitoring capabilities.
- **External Monitoring Solutions**: Integrating Prometheus for metric collection and Grafana for visual analytics allows for extensive monitoring and real-time performance tracking of PostgreSQL environments.

Backup and Restore

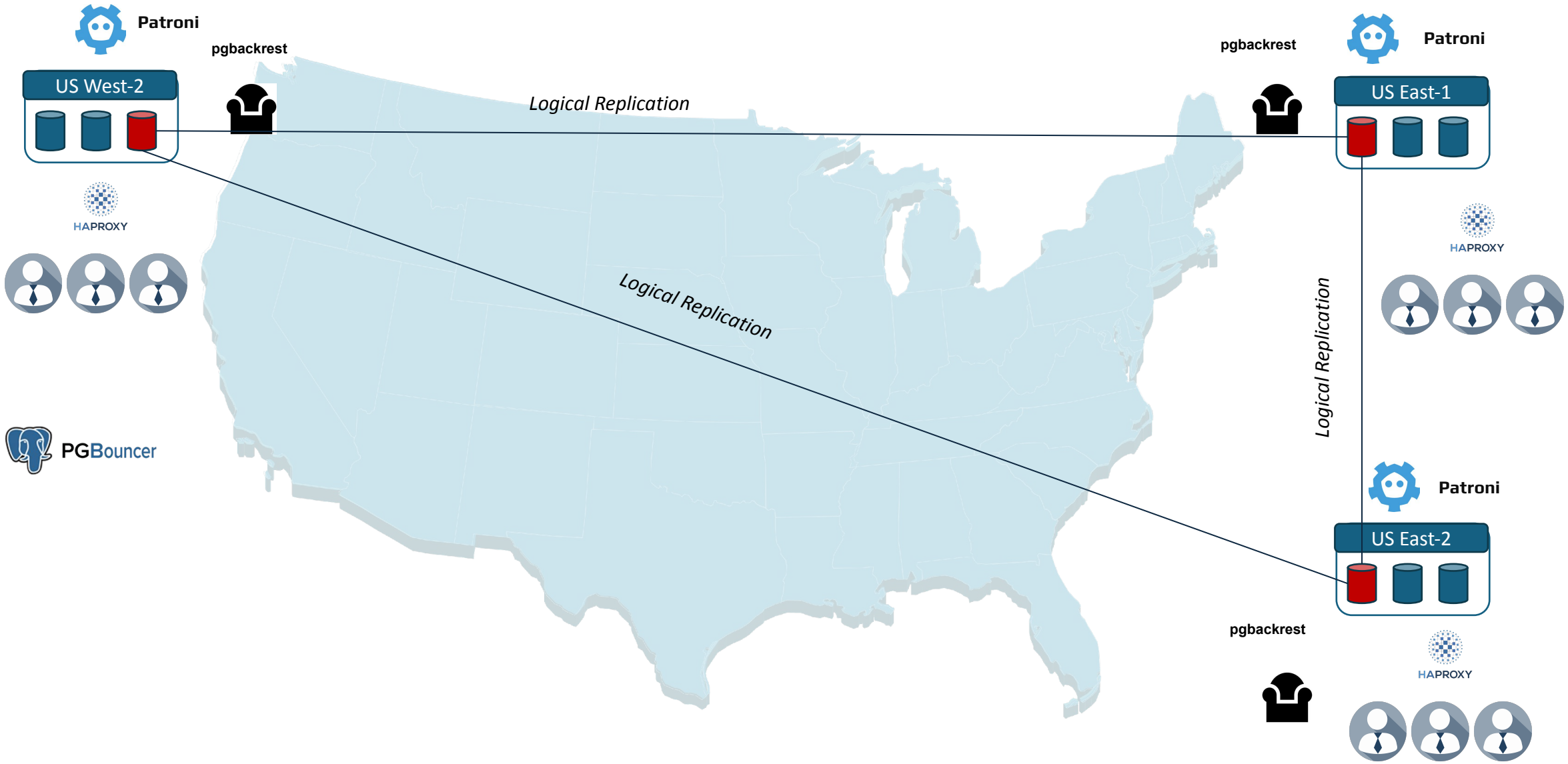
- **pg_dump and pg_dumpall:** These are the primary tools for backing up PostgreSQL. pg_dump is used for backing up individual databases, while pg_dumpall is useful for backing up all databases on a server, including global objects like roles and tablespaces.
- **Barman**: This is a popular third-party management tool for disaster recovery of PostgreSQL databases. Barman allows remote backups, providing point-in-time recovery and integration with streaming replication.
- **pgBackRest**: Another robust tool that offers features like incremental backups, parallel processing for faster backup and restore times, and on-the-fly compression and encryption to enhance security and reduce storage requirements.
- **WAL-E**: A tool designed for managing continuous archiving of PostgreSQL WAL files. WAL-E supports storing backups in cloud storage services like AWS S3, Google Cloud Storage, and Azure Blob Storage, facilitating disaster recovery.
- **Continuous Archiving and Point-in-Time Recovery (PITR)**: PostgreSQL supports continuous archiving of transaction logs (WAL files), which allows for precise point-in-time recovery. Tools like pg_basebackup and configurations in the postgresql.conf can be utilized to set up and manage WAL shipping for robust data protection and recovery.

Backup and Restore



- **Incremental Backups**: pgBackRest supports incremental backups, which only capture changes made since the last backup, significantly reducing the amount of data transferred and storage space required.
- **Parallel Processing**: It utilizes parallel processing to expedite the backup and restore processes, making it highly efficient for handling large databases by using multiple CPU cores to manage the workload.
- **Point-in-Time Recovery** (PITR): pgBackRest allows for point-in-time recovery, enabling administrators to restore a database to a specific moment, which is crucial for minimizing data loss in case of an error or issue occurring post the last full backup.
- **Compression**: It provides on-the-fly compression to save storage space and supports encryption of backup data, enhancing the security of data stored in backup files.
- **Remote Backup Capabilities**: pgBackRest can perform backups from a remote server, reducing the load on the primary database server and allowing more flexible and robust backup architecture configurations.

High Availability in PostgreSQL



Questions

Code is like clay; in the hands of a skilled craftsman, it can be molded into something that stands the test of time. Remember, the art is not in writing code, but in crafting solutions that endure. Let's build not just for today, but for the future.

Ibrar Ahmed

pgEdge