Database Migration

- Database Migration Service for PostgreSQL
- Cloning a Service to Another Provider or Region
- Manual Migration Using pg_dump and pg_restore

Database Migration Service for PostgreSQL

Elestio provides a structured approach for migrating PostgreSQL databases from various environments, such as on-premises systems or other cloud platforms, to its managed services. This process ensures data integrity and minimizes downtime, facilitating a smooth transition to a managed environment.

Key Steps in Migrating to Elestio

Pre-Migration Preparation

Before initiating the migration process, it's essential to undertake thorough preparation to ensure a smooth transition:

- **Create an Elestio Account:** Register on the Elestio platform to access their suite of managed services. This account will serve as the central hub for managing your PostgreSQL instance and related resources.
- **Deploy the Target PostgreSQL Service:** Set up a new PostgreSQL instance on Elestio to serve as the destination for your data. It's crucial to match the software version of your current PostgreSQL database to avoid compatibility issues during data transfer. Detailed prerequisites and guidance can be found in Elestio's migration documentation.

Initiating the Migration Process

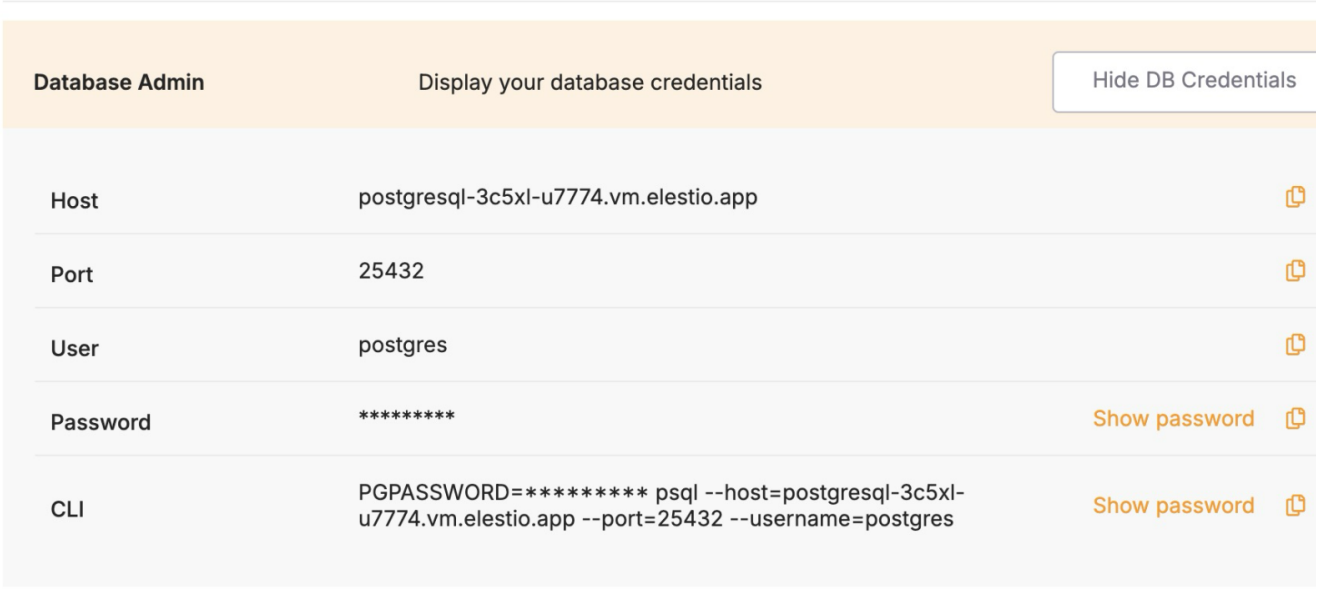
With the preparatory steps completed, you can proceed to migrate your PostgreSQL database to Elestio:

1. **Access the Migration Tool:** Navigate to the overview of your PostgreSQL service on the Elestio dashboard. Click on the "Migrate Database" button to initiate the migration process. This tool is designed to facilitate a smooth transition by guiding you through each step.
2. **Configure Migration Settings:** A modal window will open, prompting you to ensure that your target service has sufficient disk space to accommodate your database. Adequate

storage is vital to prevent interruptions during data transfer. Once confirmed, click on the "Get started" button to proceed.

3. **Validate Source Database Connection:** Provide the connection details for your existing PostgreSQL database, including:
   - **Hostname:** The address of your current database server.
   - **Port:** The port number on which your PostgreSQL service is running (default is 5432).
   - **Database Name:** The name of the database you intend to migrate.
   - **Username:** The username with access privileges to the database.
   - **Password:** The corresponding password for the user.

   After entering these details, click on "Run Check" to validate the connection. This step ensures that Elestio can securely and accurately access your existing data. You can find these details under Database admin section under your deployed PostgreSQL service.



4. **Execute the Migration:** If all checks pass without errors, initiate the migration by selecting "Start migration." Monitor the progress through the real-time migration logs displayed on the dashboard. This transparency allows for immediate detection and resolution of any issues, ensuring data integrity throughout the process.

Post-Migration Validation and Optimization

After completing the migration, it's crucial to perform validation and optimization tasks to ensure the integrity and performance of your database in the new environment:

- **Verify Data Integrity:** Conduct thorough checks to ensure all data has been accurately transferred. This includes comparing row counts, checksums, and sample data between the source and target databases. Such verification maintains the reliability of your database and ensures that no data was lost or altered during migration.
- **Test Application Functionality:** Ensure that applications interacting with the database function correctly in the new environment. Update connection strings and configurations as necessary to reflect the new database location. This step prevents potential disruptions and ensures seamless operation of dependent systems.

- **Optimize Performance:** Utilize Elestio's managed service features to fine-tune database performance. Set up automated backups to safeguard your data, monitor resource utilization to identify and address bottlenecks, and configure scaling options to accommodate future growth. These actions contribute to improved application responsiveness and overall system efficiency.
- **Implement Security Measures:** Review and configure security settings to protect your data within the Elestio environment. Set up firewalls to control access, manage user access controls to ensure only authorized personnel can interact with the database, and enable encryption where applicable to protect data at rest and in transit. Implementing these security measures safeguards your data against unauthorized access and potential threats.

Benefits of Using Elestio for PostgreSQL

Migrating your PostgreSQL database to Elestio offers several advantages:

- **Simplified Management:** Elestio automates database maintenance tasks, including software updates, backups, and system monitoring, reducing manual work. The platform provides a dashboard with real-time insights into database performance and resource usage. It allows for adjusting service plans, scaling CPU and RAM as needed. Users can modify environment variables and access software information to manage configurations.

- **Security:** Elestio keeps PostgreSQL instances updated with security patches to protect against vulnerabilities. The platform automates backups to ensure data integrity and availability. It provides secure access mechanisms, including randomly generated passwords for database instances, which can be managed through the dashboard.

- **Performance:** Elestio configures PostgreSQL instances for performance based on workload requirements. The platform supports the latest PostgreSQL versions, incorporating updates that improve database operations. Its infrastructure handles different workloads and maintains performance during high usage periods.

- **Scalability:** Elestio's PostgreSQL service allows for scaling database resources to handle growth and changing workloads without major downtime. Users can upgrade or downgrade service plans, adjusting CPU and RAM as needed. The platform supports adding network volumes to increase storage capacity.

Cloning a Service to Another Provider or Region

Migrating or cloning services across cloud providers or geographic regions is a critical part of modern infrastructure management. Whether you're optimizing for latency, preparing for disaster recovery, meeting regulatory requirements, or simply switching providers, a well-planned migration ensures continuity, performance, and data integrity. This guide outlines a structured methodology for service migration, applicable to most cloud-native environments.

Pre-Migration Preparation

Before initiating a migration, thorough planning and preparation are essential. This helps avoid unplanned downtime, data loss, or misconfiguration during the move:

- **Evaluate the Current Setup**: Begin by documenting the existing service's configuration. This includes runtime environments (container images, platform versions), persistent data (databases, object storage), network rules (ports, firewalls), and application dependencies (APIs, credentials, linked services).
- **Define the Migration Target**: Choose the new cloud provider or region you plan to migrate to. Confirm service compatibility, resource limits, and geographic latency requirements. If you're replicating an existing environment, make sure the target region supports the same compute/storage features and versions.
- **Provision the Target Environment**: Set up the target infrastructure where the service will be cloned. This could involve creating new Kubernetes clusters, VM groups, container registries, databases, or file storage volumes—depending on your stack.
- **Backup the Current Service**: Always create a full backup or snapshot of the current service and its associated data before proceeding. This acts as a rollback point in case of migration issues and ensures recovery in the event of failure.

Cloning Execution

The first step in executing a clone is to replicate the configuration of the original service in the target environment. This involves deploying the same container image or service binary using the same runtime settings. If you're using Kubernetes or container orchestrators, this can be done via Helm charts or declarative manifests. Pay close attention to environment variables, secrets,

mounted paths, storage class definitions, and health check configurations to ensure a consistent runtime environment.

Next, you'll need to migrate any persistent data tied to the service. For PostgreSQL databases, this might involve using pg_dump to export the schema and data, followed by psql or pg_restore to import it into the new instance. In more complex cases, tools like pgBackRest, wal-g, or logical replication can be used to minimize downtime during the switchover. For file-based storage, tools like rsync or rclone are effective for copying volume contents over SSH or cloud storage backends. It's crucial to verify compatibility across disk formats, database versions, and encoding standards to avoid corruption or mismatched behavior.

After replicating the environment and data, it's important to validate the new service in isolation. This means confirming that all application endpoints respond as expected, background tasks or cron jobs are functioning, and third-party integrations (e.g., payment gateways, S3 buckets) are accessible. You should test authentication flows, data read/write operations, and retry logic to ensure the new service is functionally identical. Use observability tools to monitor resource consumption and application logs during this stage.

Once validation is complete, configure DNS and route traffic to the new environment. This might involve updating DNS A or CNAME records, changing cloud load balancer configurations, or applying new firewall rules. For high-availability setups, consider using health-based routing or weighted DNS to gradually transition traffic from the old instance to the new one.

Post-Migration Validation and Optimization

Once the new environment is live and receiving traffic, focus on optimizing and securing the setup:

- **Validate Application Functionality**: Test all integrations, user workflows, and background jobs to confirm proper behavior. Review logs for silent errors or timeouts. Ensure all applications pointing to the service are updated with the new URL or connection string.
- **Monitor Performance**: Analyze load, CPU, memory, and storage utilization. Scale resources as needed, or optimize runtime settings for the new provider/region. Enable autoscaling where applicable.
- **Secure the Environment**: Implement firewall rules, IP restrictions, and access controls. Rotate secrets and validate that no hardcoded credentials or endpoints point to the old service.
- **Cleanup and Documentation**: Once validated, decommission the old setup safely. Update internal documentation with new deployment details, endpoint addresses, and any configuration changes.

Benefits of Cloning

Cloning a database service, particularly for engines like PostgreSQL offers several operational and strategic advantages. It allows teams to test schema migrations, version upgrades, or major application features in an isolated environment without affecting production. By maintaining a cloned copy, developers and QA teams can work against realistic data without introducing risk.

Cloning also simplifies cross-region redundancy setups. A replica in another region can be promoted quickly if the primary region experiences an outage. For compliance or analytics purposes, cloned databases allow for read-only access to production datasets, enabling safe reporting or data processing without interrupting live traffic.

Additionally, rather than building a new environment from scratch, you can clone the database into another provider, validate it, and cut over with minimal disruption. This helps maintain operational continuity and reduces the effort needed for complex migrations.

Manual Migration Using pg_dump and pg_restore

Manual Migrations using PostgreSQL's built-in tools `pg_dump` and `pg_restore` are ideal for users who prefer full control over data export and import, particularly during provider transitions, database version upgrades, or when importing an existing self-managed PostgreSQL dataset into Elestio's managed environment. This guide walks through the process of performing a manual migration to and from Elestio PostgreSQL services using command-line tools, ensuring that your data remains portable, auditable, and consistent.

When to Use Manual Migration

Manual migration using `pg_dump` and `pg_restore` is well-suited for scenarios where full control over the data export and import process is required. This method is particularly useful when migrating from an existing PostgreSQL setup, whether self-hosted, on-premises, or on another cloud provider, into Elestio's managed PostgreSQL service. It allows for one-time imports without requiring continuous connectivity between source and target systems.

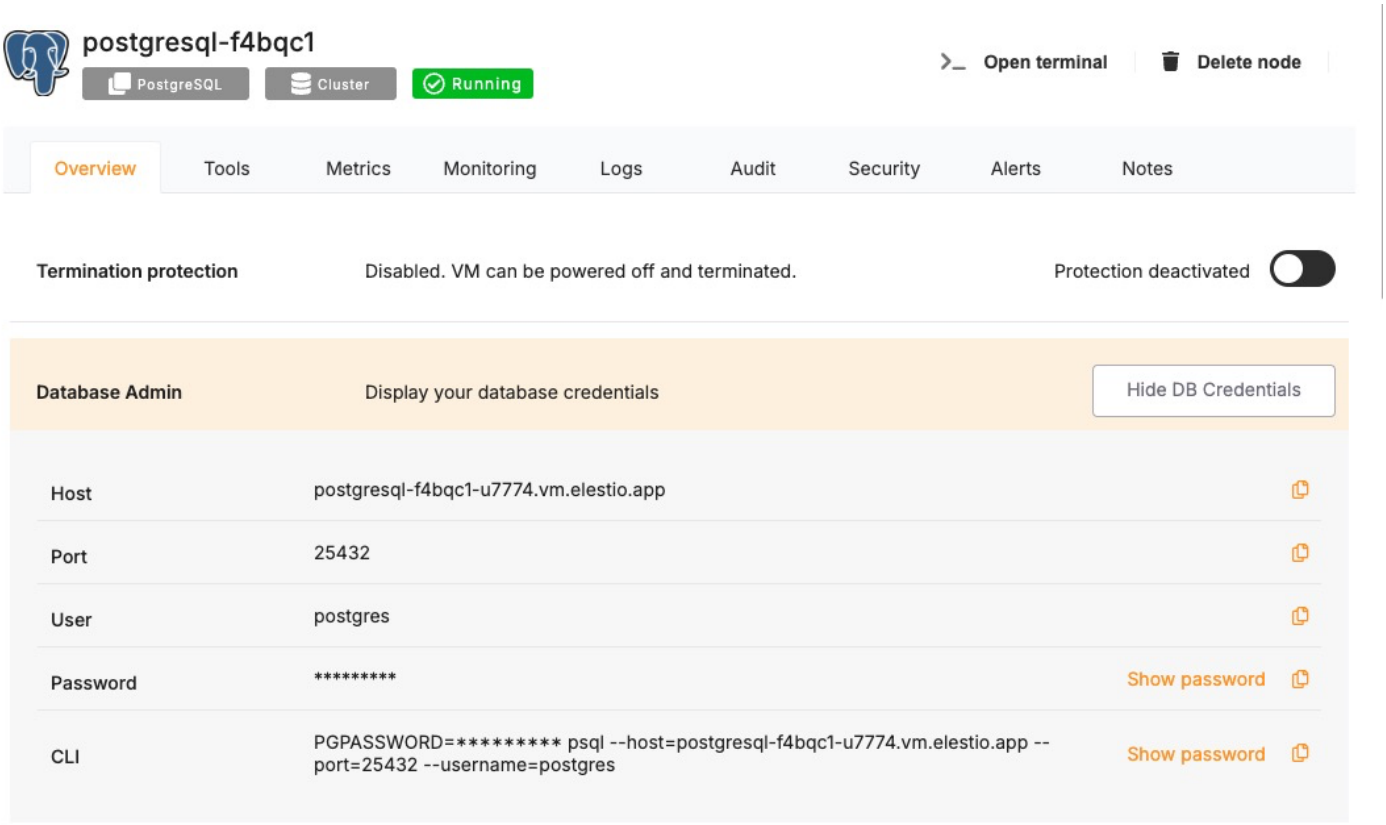
This approach is also ideal when dealing with version upgrades, as PostgreSQL's logical backups can be restored into newer versions without compatibility issues. In situations where Elestio's built-in snapshot or replication tools aren't applicable such as migrations from isolated environments or selective schema transfers, manual migration becomes the most practical option. Additionally, this method enables users to retain portable, versioned backups outside of Elestio's infrastructure, which can be archived, validated offline, or re-imported into future instances.

Performing the Migration

Prepare the Environments

Before initiating a migration, verify that PostgreSQL is properly installed and configured on both the source system and your Elestio service. On the source, you need an active PostgreSQL instance with a user account that has sufficient privileges to read schemas, tables, sequences, and any installed extensions. The user must also be allowed to connect over TCP if the server is remote.

On the Elestio side, provision a PostgreSQL service from the dashboard. Once deployed, retrieve the connection information from the Database admin tab. This includes the hostname, port, database name, username, and password. You'll use these credentials to connect during the restore step. Ensure that your IP is allowed to connect under the Cluster Overview > Security > Limit access per IP section; otherwise, the PostgreSQL port will be unreachable during the migration.



Create a Dump Using pg_dump

In this step, you generate a logical backup of the source database using `pg_dump`. This utility connects to the PostgreSQL server and extracts the structure and contents of the specified database. It serializes tables, indexes, constraints, triggers, views, and functions into a consistent snapshot. The custom format (-Fc) is used because it produces a compressed binary dump that can be restored selectively using pg_restore.

```
pg_dump -U <source_user> -h <source_host> -p <source_host> -Fc <source_database> > backup.dump
```

This command connects to the source server (-h), authenticates with the user (-U), targets the database (source_database), and exports the entire schema and data into `backup.dump`. The resulting file is portable and version-aware. You can also add `--no-owner` and `--no-acl` If you're migrating between environments that use different database roles or access models. This prevents restore-time errors related to ownership mismatches.

Transfer the Dump File to the Target

If your source and target environments are on different hosts, the dump file must be transferred securely. This step ensures the logical backup is available on the system from which you'll perform the restore. You can use secure copy (scp), rsync, or any remote file transfer method.

```
scp backup.dump your_user@your_workstation:/path/to/local/
```

If restoring from your local machine to Elestio, ensure the dump file is stored in a location readable by your current shell user. Elestio does not require the file to be uploaded to its servers; the restore is performed by connecting over the network using standard PostgreSQL protocols. At this point, your backup is isolated from the source environment and ready for import.

Create the Target Database

By default, Elestio provisions a single database instance. However, if you wish to restore into a separate database name or if your dump references a different name, you must create the new database manually. Use the psql client to connect to your Elestio service using the credentials from the dashboard.

```
psql -U <elestio_user> -h <elestio_host> -p <elestio_host> -d postgres
```

Within the psql session, create the database:

```
CREATE DATABASE target_database WITH ENCODING='UTF8' LC_COLLATE='en_US.UTF-8'
LC_CTYPE='en_US.UTF-8' TEMPLATE=template0;
```

This ensures that the new database has consistent encoding and locale settings, which are critical for text comparison, sorting, and indexing. Using template0 avoids inheriting default extensions or templates that might conflict with your dump file. At this stage, you can also create any roles, schemas, or extensions that were used in the original database if they are not included in the dump.

Restore Using pg_restore

With the target database created and the dump file in place, initiate the restoration using `pg_restore`. This tool reads the custom-format archive and reconstructs all schema and data objects in the new environment.

```
pg_restore -U elestio_user -h elestio_host -p 5432 -d target_database -Fc /path/to/backup.dump
--verbose
```

This command establishes a network connection to the Elestio PostgreSQL service and begins issuing `CREATE`, `INSERT`, and `ALTER` statements to rebuild the database. The `--verbose` flag provides

real-time feedback about the objects being restored. You can also use `--jobs=N` to run the restore in parallel, improving performance for large datasets, provided the dump was created with `pg_dump --jobs=N`.

It’s important to ensure that all referenced extensions, collations, and roles exist on the target instance to avoid partial restores. If errors occur, the logs will point to the missing components or permission issues that need to be resolved.

Validate the Migration

Once the restore completes, you must validate the accuracy and completeness of the migration. Connect to the Elestio database using psql or a PostgreSQL GUI (such as pgAdmin or TablePlus), and run checks across critical tables.

Begin by inspecting the table existence and row counts:

```
\dt
SELECT COUNT(*) FROM your_important_table;s
```

Validate views, functions, and indexes, especially if they were used in reporting or application queries. Run application-specific health checks, reinitialize ORM migrations if applicable, and confirm that the application can read and write to the new database without errors.

If you made any changes to connection strings or credentials, update your environment variables or secret managers accordingly. Elestio also supports automated backups, which you should enable post-migration to protect the restored dataset.

Benefits of Manual Migration

Manual PostgreSQL migration using `pg_dump` and `pg_restore` on Elestio provides several key advantages:

- **Compatibility and Portability**: Logical dumps allow you to migrate from any PostgreSQL-compatible source into Elestio, including on-premises systems, Docker containers, or other clouds.
- **Version-Safe Upgrades**: The tools support migrating across PostgreSQL versions, which is ideal during controlled upgrades.
- **Offline Archiving**: Manual dumps serve as portable archives for cold storage, disaster recovery, or historical snapshots.
- **Platform Independence**: You retain full access to PostgreSQL’s native tools without being locked into Elestio-specific formats or interfaces.

This method complements Elestio’s automated backup and migration features by enabling custom workflows and one-off imports with full visibility into each stage.