PostgreSQL Migration Tool - User Guide

Overview

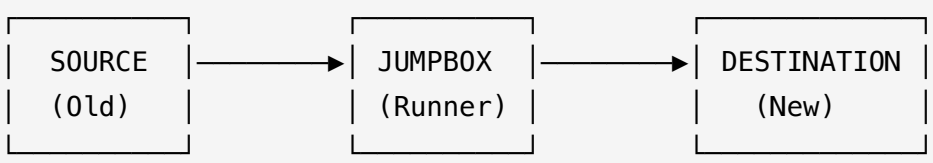
The IBP Migration Tool (pg-migrator.sh) provides a comprehensive solution for migrating IBP EC2 instances from older Ubuntu versions with PostgreSQL 12 to Ubuntu 22.04 with PostgreSQL 14. The tool orchestrates the entire migration process from a jumpbox server, managing SSH connections to both source and destination servers while handling database dumps, file transfers, and system configuration synchronization.

Purpose

This tool automates the complete migration workflow including PostgreSQL database dumps and restores with parallel processing, server configuration file transfers covering jetty configurations, SSH keys, salt minion settings, and custom scripts. It also handles optional bi_cube environment setup, system configuration synchronization for timezone and hostname settings, and comprehensive data validation and integrity checks throughout the process.

Architecture

The migration follows a three-tier architecture where the jumpbox acts as an intermediary orchestrator. The source server (running the old environment) connects to the jumpbox (the runner), which then connects to the destination server (the new environment). This design ensures controlled data flow and allows for validation at each stage.



Prerequisites

Jumpbox Requirements

The jumpbox must run Bash 4.0 or higher and have SSH access configured to both source and destination servers. Essential utilities include rsync and zstd for file transfers and compression. Ensure sufficient disk space in /tmp for temporary files during the migration process.

Source Server Requirements

The source server should be running PostgreSQL 12 on port 27095 (default configuration). The smoothie user must have sudo privileges, and the stopdw and startdw commands must be available for service management. A minimum of 10GB free disk space is required for database dumps and temporary files.

Destination Server Requirements

The destination server must be running Ubuntu 22.04 with PostgreSQL 14 installed and running on port 27095. The smoothie user requires sudo privileges, and at least 10GB of free disk space must be available for receiving the migration data.

SSH Configuration

Passwordless SSH must be configured from the jumpbox to both servers using properly configured SSH keys. The user account must have sudo privileges on both servers to execute administrative commands during the migration process.

Configuration

The script accepts configuration through environment variables, providing flexibility for different deployment scenarios. Required variables include SOURCE_HOST for the source server hostname and DEST_HOST for the destination server hostname.

Optional configuration variables include SOURCE_PORT and DEST_PORT (both default to 27095), SOURCE_SSH_USER and DEST_SSH_USER (both default to smoothie), BACKUP_DIR (defaults to /tmp/pg_migration/dumps), SERVER_FILES_BACKUP_DIR (defaults to /tmp/pg_migration/server_files), and PARALLEL_JOBS (defaults to 2).

```
export SOURCE_HOST="source-server-hostname"
export DEST_HOST="destination-server-hostname"
export SOURCE_PORT="27095"
export DEST_PORT="27095"
export SOURCE_SSH_USER="smoothie"
export DEST_SSH_USER="smoothie"
export BACKUP_DIR="/tmp/pg_migration/dumps"
export SERVER_FILES_BACKUP_DIR="/tmp/pg_migration/server_files"
export PARALLEL_JOBS="2"
```

If environment variables are not set, the script will prompt interactively for required values during execution.

Usage

Basic Execution

You can run the script with environment variables pre-configured or allow it to prompt interactively for required information.

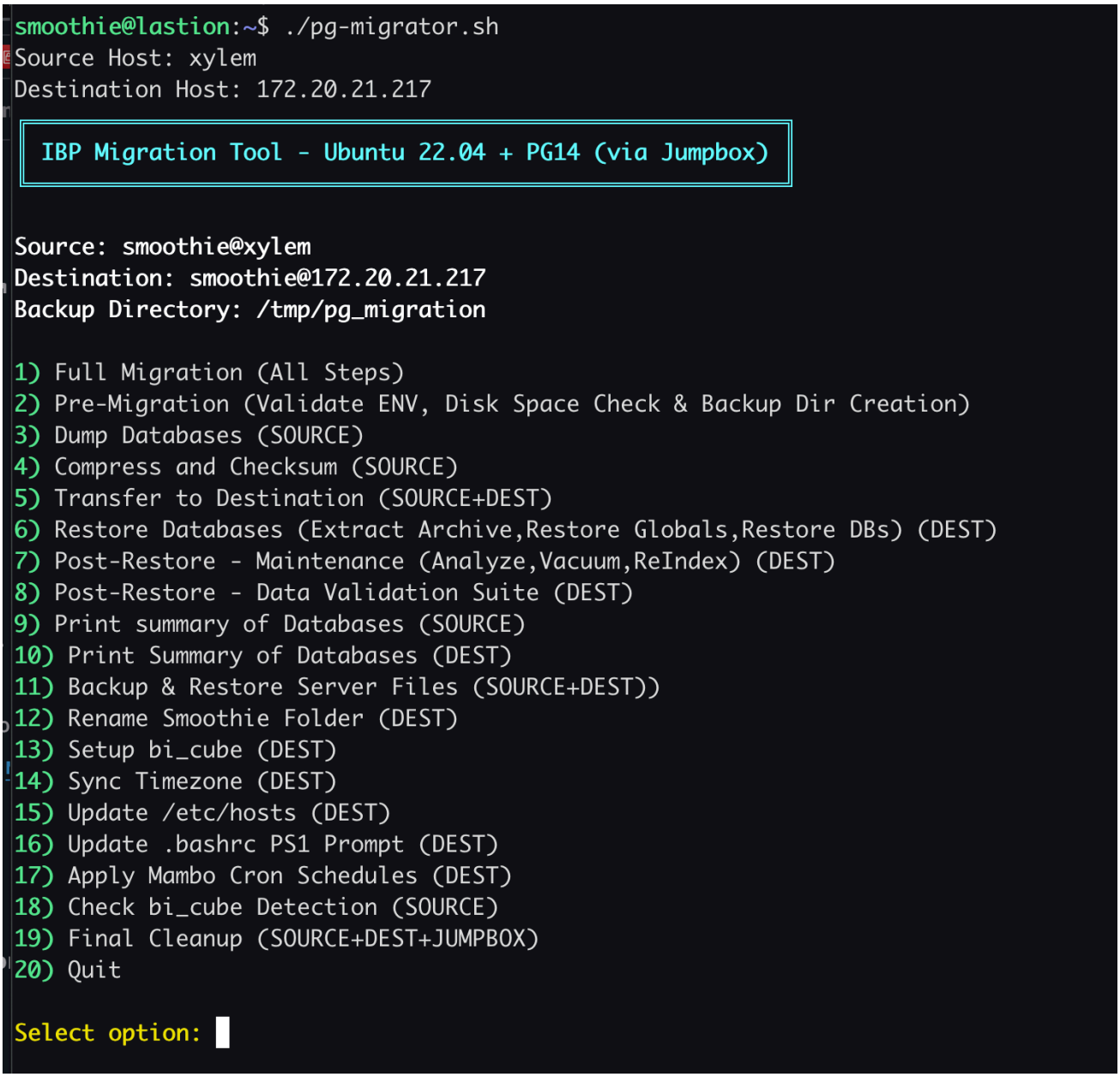
```
SOURCE_HOST=old-server DEST_HOST=new-server ./pg-migrator.sh
```

For interactive mode, simply execute the script without environment variables and respond to the prompts.

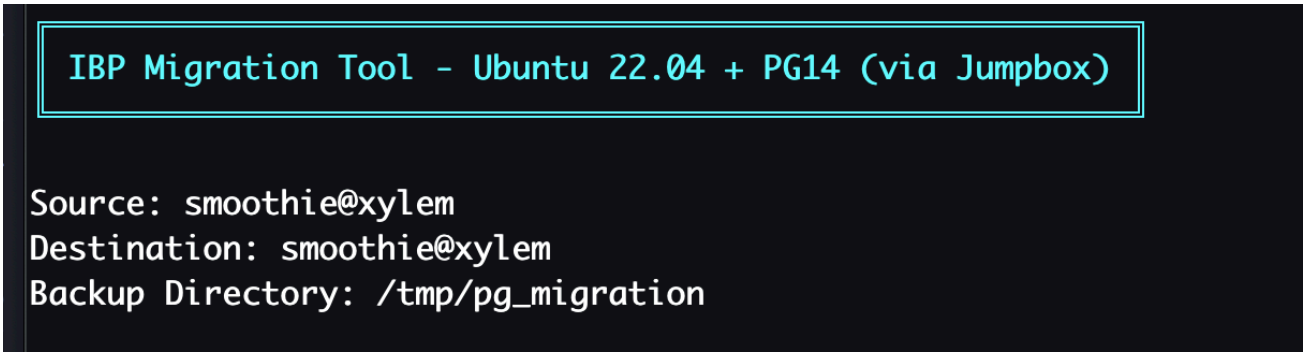
```
./pg-migrator.sh
```

Menu System

The script presents an interactive menu with 20 distinct options, allowing you to execute the complete migration workflow or individual steps as needed.



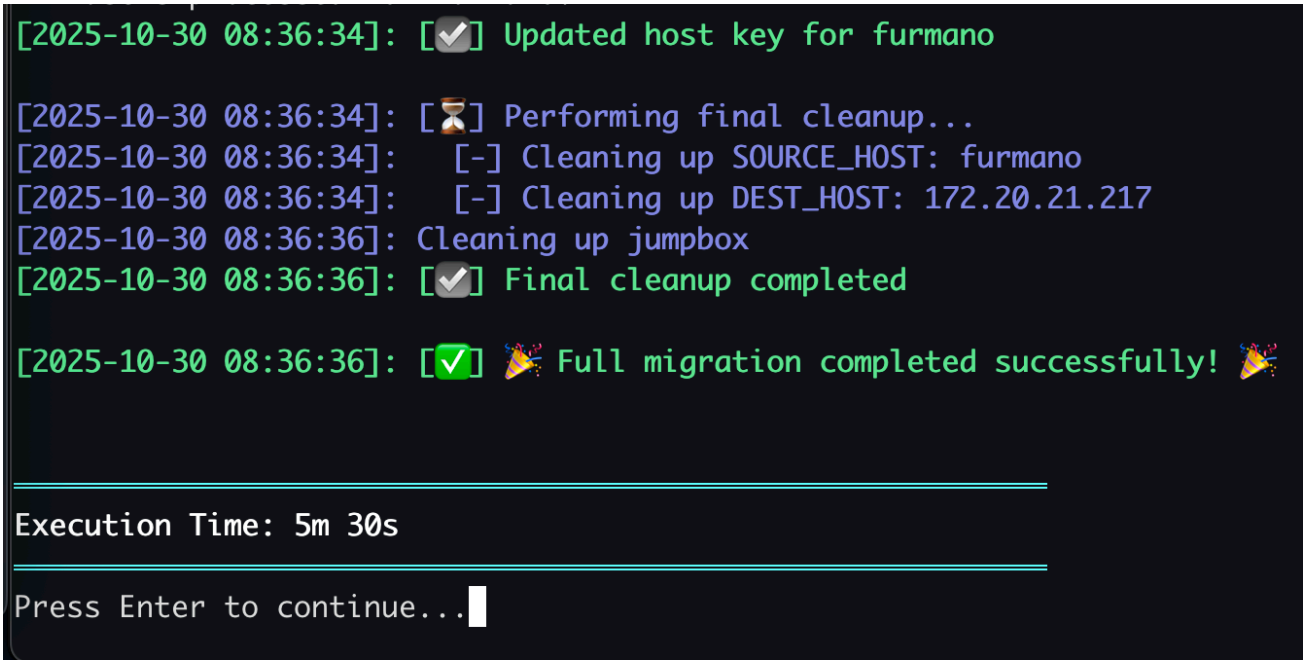
The menu displays the current source and destination configuration at the top, making it easy to verify you're working with the correct servers before proceeding.



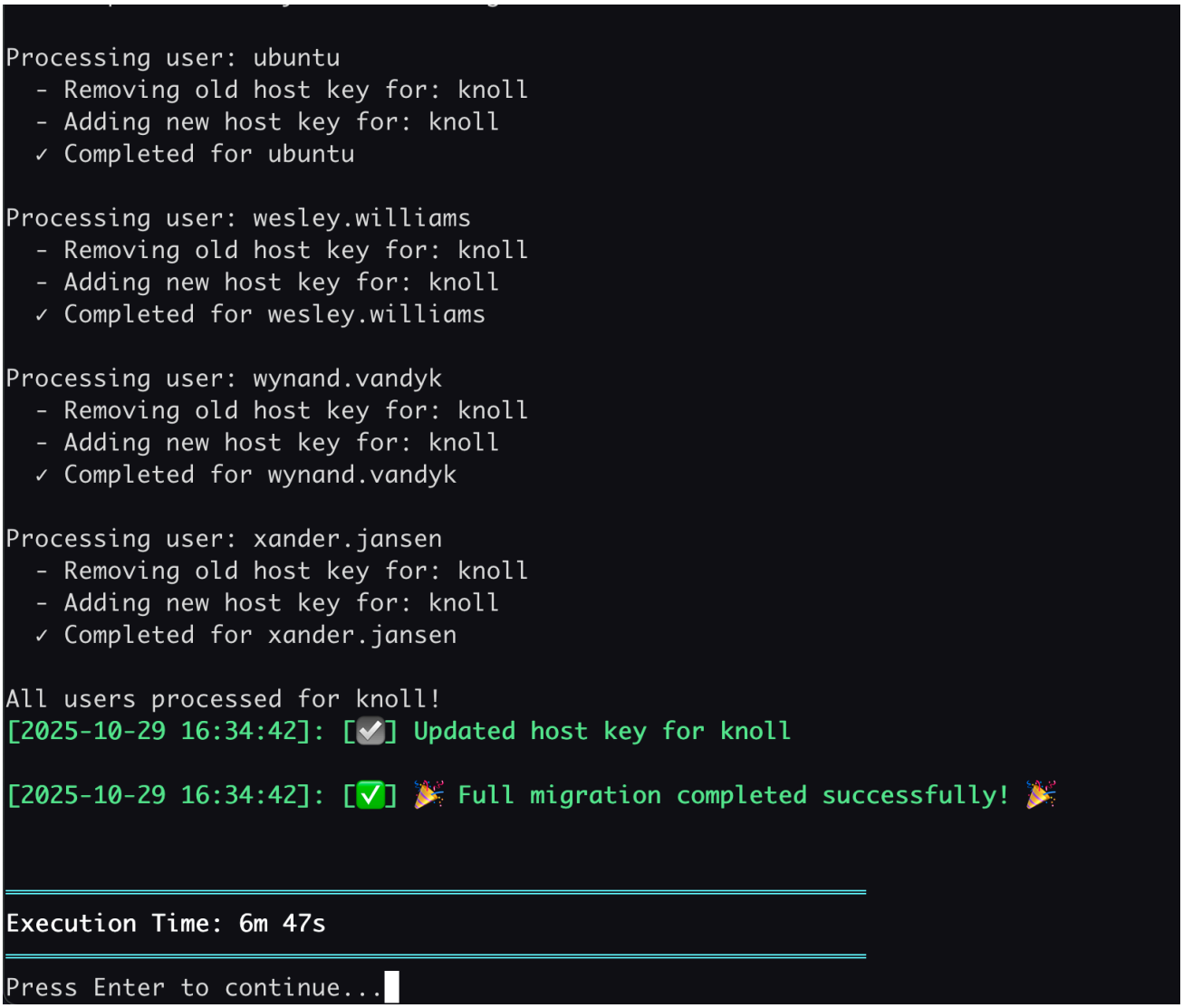
Menu Options Explained

Option 1: Full Migration (All Steps)

This option executes the complete end-to-end migration workflow automatically, running through all phases from validation to cleanup. It's ideal for straightforward migrations where all prerequisites have been verified.

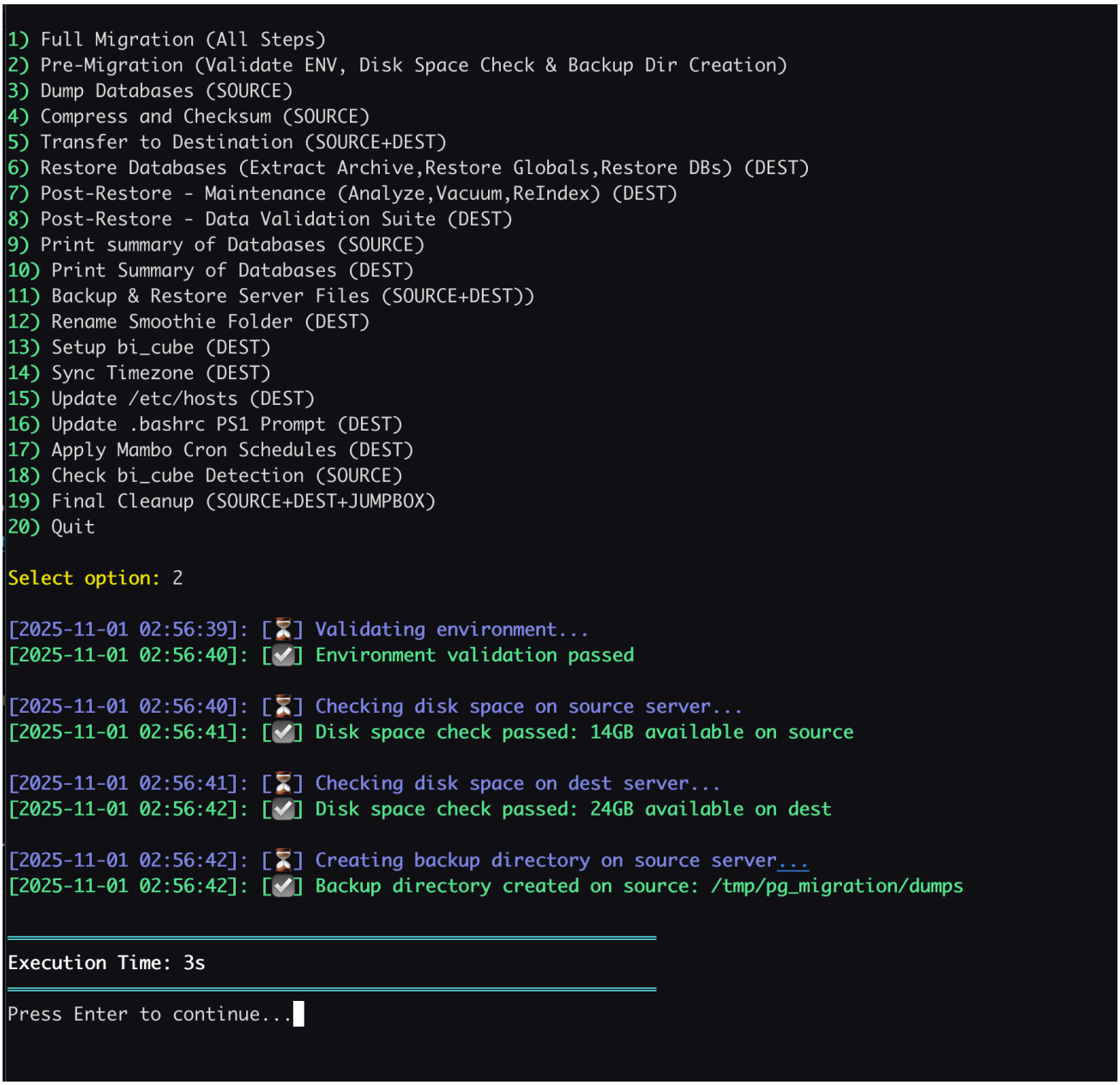


The full migration process displays detailed timing metrics for each phase, allowing you to monitor progress and identify any bottlenecks.



Option 2: Pre-Migration

The pre-migration phase validates SSH connectivity to both servers, checks available disk space to ensure sufficient room for dumps and transfers, creates necessary backup directories, and detects whether bi_cube is installed on the source server.



Option 3: Dump Databases (SOURCE)

This option stops DW services on the source server, applies PostgreSQL maintenance settings optimized for dump operations, exports global objects including roles and tablespaces, dumps all user databases in parallel for efficiency, and reverts maintenance settings after completion.

Option 4: Compress and Checksum (SOURCE)

Creates a compressed tar archive using zstd compression, includes both database dumps and server configuration files, and generates checksums for transfer validation.

Option 5: Transfer to Destination

Transfers the archive from source to jumpbox, then from jumpbox to destination, and validates checksums to ensure data integrity during transfer.

Option 6: Restore Databases (DEST)

Extracts the archive on the destination server, sets maintenance settings optimized for restore operations, restores global objects first, then restores all databases in parallel, and reverts maintenance settings after completion.

Option 7: Post-Restore - Maintenance (DEST)

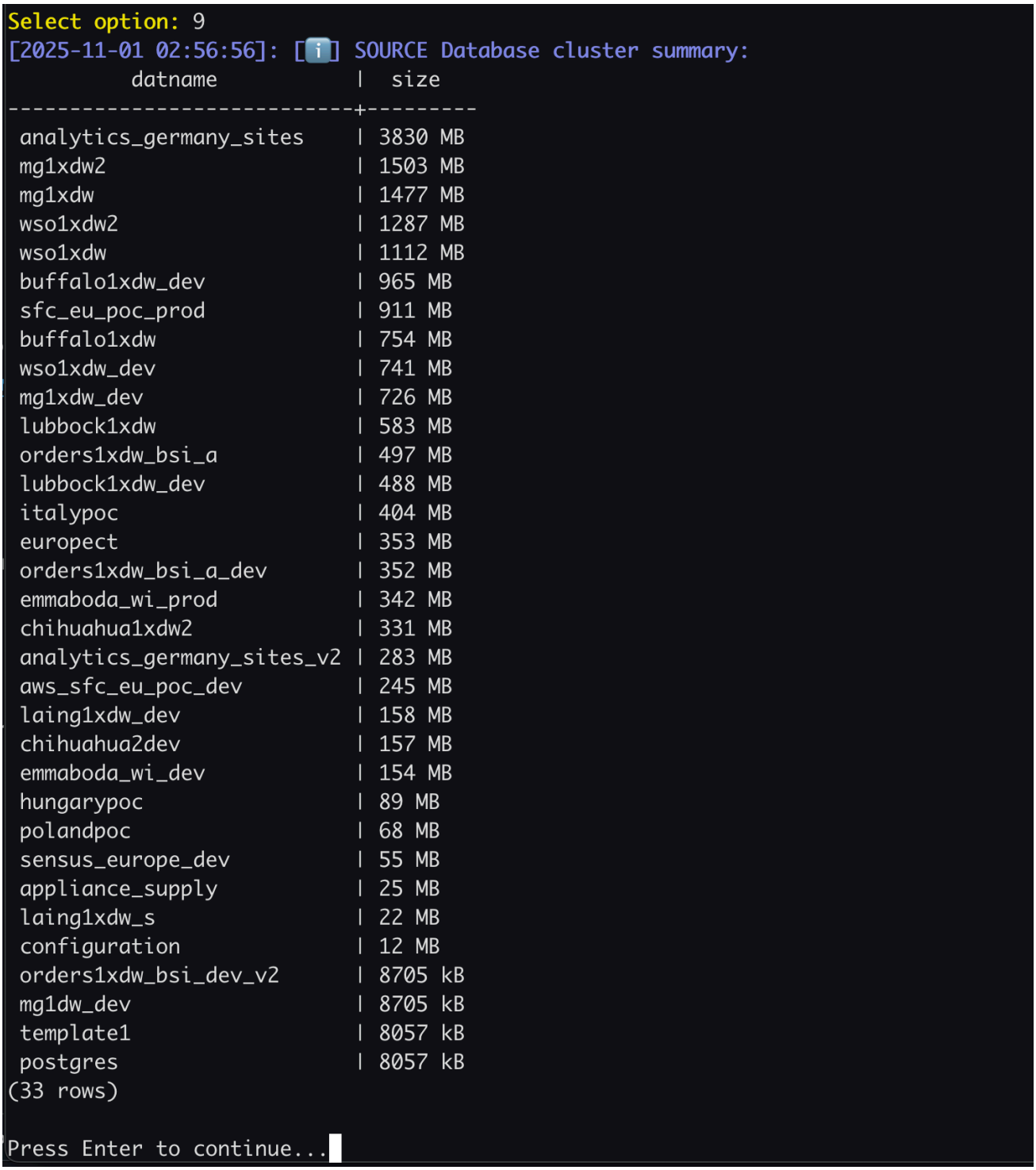
Runs ANALYZE on all databases to update statistics, executes VACUUM ANALYZE to reclaim space and update statistics, and rebuilds all indexes using REINDEX for optimal performance.

Option 8: Post-Restore - Data Validation Suite (DEST)

Validates row counts to ensure all data was transferred, checks constraints to verify referential integrity, and validates extensions to confirm all PostgreSQL extensions are properly installed.

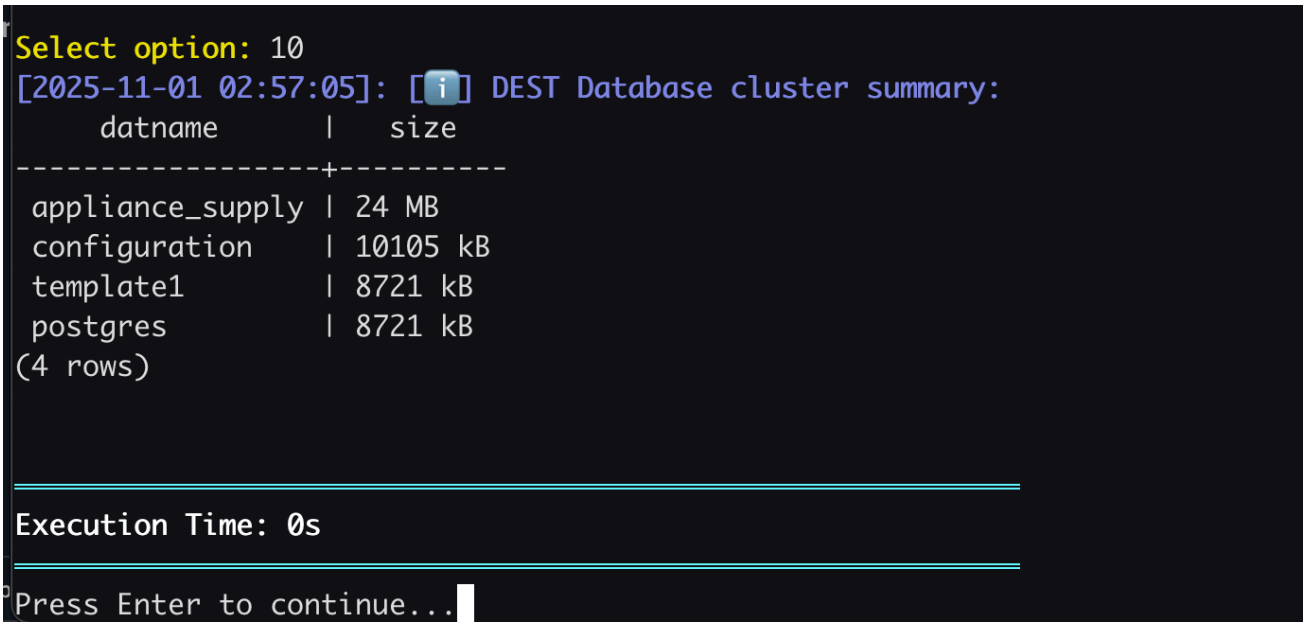
Option 9: Print Summary of Databases (SOURCE)

Displays database sizes on the source server, providing a baseline for comparison after migration.



Option 10: Print Summary of Databases (DEST)

Displays database sizes on the destination server, allowing you to verify that all databases were restored correctly.



Option 11: Backup & Restore Server Files

Renames the existing smoothie folder to preserve the original installation and restores configuration files from the archive to their proper locations.

Option 12: Rename Smoothie Folder (DEST)

Renames /opt/smoothie11 to /opt/smoothie11_old, preserving the original installation before restoring files from the source server.

Option 13: Setup bi_cube (DEST)

If bi_cube is detected on the source server, this option configures the bi_cube environment on the destination, sets appropriate file ownership, creates a Python virtual environment, and installs required packages including boto3, mysql-connector-python, psycopg2-binary, and privatebinapi.

Option 14: Sync Timezone (DEST)

Synchronizes the timezone configuration from source to destination, ensuring consistent time handling across the migrated system.

Option 15: Update /etc/hosts (DEST)

Updates the hostname and hosts file on the destination server to match the source configuration.

Option 16: Update .bashrc PS1 Prompt (DEST)

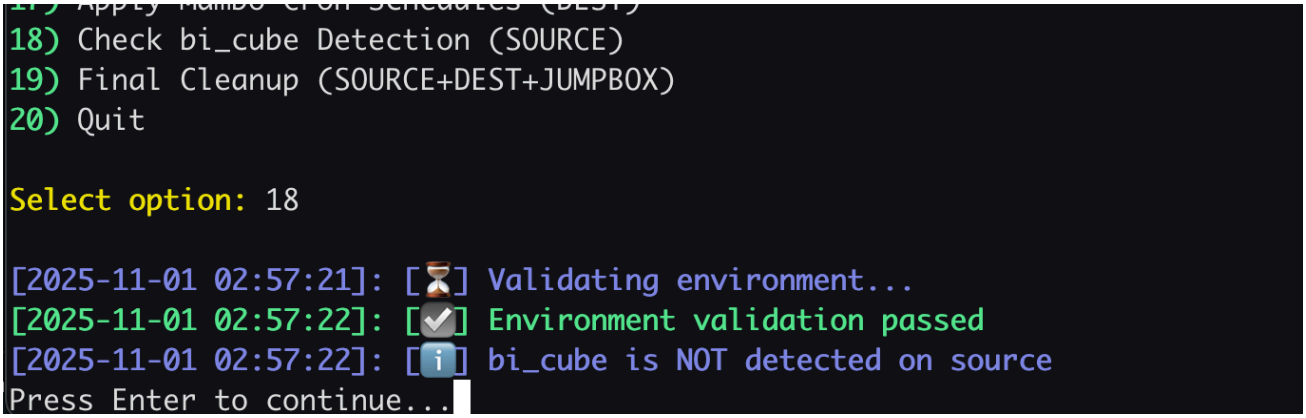
Updates the bash prompt to reflect the correct hostname, providing clear visual feedback about which server you're working on.

Option 17: Apply Mambo Cron Schedules (DEST)

Applies crontab schedules using UpdateSchedule.sh, ensuring scheduled tasks are properly configured on the destination server.

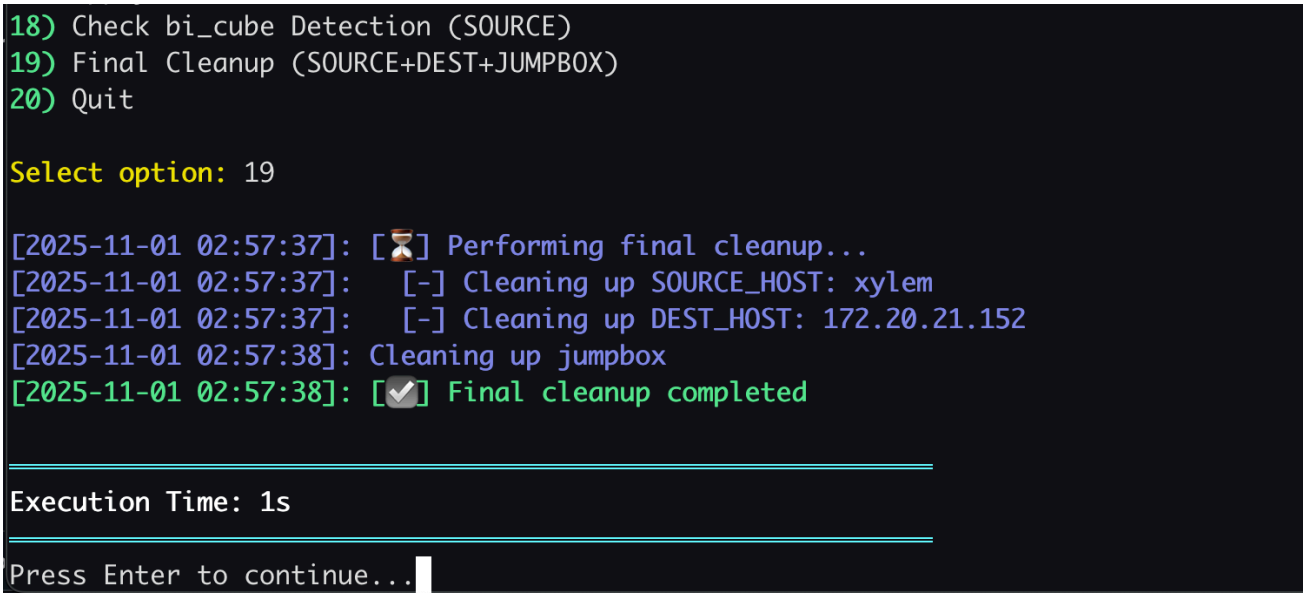
Option 18: Check bi_cube Detection (SOURCE)

Checks whether bi_cube is installed on the source server by looking for the presence of /etc/profile.d/ibp.sh.



Option 19: Final Cleanup

Removes temporary files from source, destination, and jumpbox servers, freeing up disk space used during the migration process.



Option 20: Quit

Exits the script gracefully, returning control to the shell.

Migration Workflow

Phase 1: Preparation

The preparation phase validates the environment and connectivity to both servers, checks available disk space to ensure sufficient room for the migration, creates necessary backup directories, and stops DW services on both servers to ensure data consistency.

Phase 2: Database Dump

During the database dump phase, the script sets PostgreSQL maintenance settings optimized for dump operations, exports global objects including roles and tablespaces, dumps all databases in parallel to minimize downtime, reverts maintenance settings to defaults, and restarts DW on the source server.

Phase 3: Archive & Transfer

The archive and transfer phase creates a compressed tar archive using zstd compression, generates checksums for validation, transfers the archive via the jumpbox, and validates checksums on the destination to ensure data integrity.

Phase 4: Database Restore

The database restore phase extracts the archive on the destination server, sets maintenance settings optimized for restore operations, restores global objects first to establish roles and permissions, restores databases in parallel for efficiency, and reverts maintenance settings after completion.

Phase 5: Post-Restore

Post-restore operations run ANALYZE to update database statistics, execute VACUUM to reclaim space and update statistics, run REINDEX to rebuild all indexes for optimal performance, and validate data integrity through multiple checks.

Phase 6: Configuration

The configuration phase renames the smoothie folder to preserve the original installation, restores server files from the archive, configures bi_cube if detected on the source server, synchronizes timezone settings, updates hostname and hosts file, updates the bash prompt, and applies cron schedules.

Phase 7: Finalization

The finalization phase displays database summaries for verification, updates host keys in the known_hosts file, and performs final cleanup of temporary files.

Performance Optimization

The script automatically optimizes PostgreSQL settings based on available system resources, ensuring efficient dump and restore operations.

CPU-Based Parallelism

Parallel workers are set to 75% of available CPU cores, and parallel jobs per database are set to 50% of half cores with a minimum of 2 jobs.

Memory-Based Settings

Maintenance work memory is set to 25% of RAM within a 2-8GB range, shared buffers are set to 25% of RAM within a 2-4GB range, and effective cache size is set to 37% of RAM with a minimum of 4GB.

WAL Settings

Checkpoint timeout is set to 1 hour, max WAL size is set to 16GB, min WAL size is set to 4GB, WAL compression is enabled, and synchronous commit is disabled during migration for performance.

Error Handling

The script includes comprehensive error handling throughout all operations. All functions return proper exit codes for programmatic checking, failed operations are logged with timestamps for troubleshooting, critical failures halt execution to prevent data corruption,

and non-critical failures issue warnings and continue execution.

Logging

All output includes colored timestamps for easy reading, operation status indicators (⏳ for in progress, ☑️ for completed, ⚠️ for warnings, ✅ for success), execution time tracking for performance monitoring, and detailed progress information throughout the migration.

Safety Features

The script includes multiple safety features to protect your data. Validation checks verify SSH connectivity and disk space before operations begin. Backup preservation ensures original data remains on the source until cleanup is explicitly executed. Incremental execution through the menu allows step-by-step execution for troubleshooting. Rollback capability automatically reverts maintenance settings if operations fail. Data validation performs multiple checks post-restore to ensure integrity.

Troubleshooting

SSH Connection Failures

Test SSH connectivity manually to identify connection issues.

```
ssh -o ConnectTimeout=5 smoothie@source-host "echo 'SSH OK'"
ssh -o ConnectTimeout=5 smoothie@dest-host "echo 'SSH OK'"
```

Insufficient Disk Space

Check available space on both servers before starting the migration.

```
df -h /tmp
```

Database Dump Failures

Check PostgreSQL logs on the source server for error messages, verify pg_dump version compatibility between source and destination, and ensure sufficient disk space is available

for dumps.

Transfer Failures

Verify network connectivity between jumpbox and both servers, check that rsync is installed and available, and ensure sufficient bandwidth for large transfers.

Restore Failures

Check PostgreSQL logs on the destination server for error messages, verify PostgreSQL 14 is running and accessible, and ensure sufficient disk space is available for restored databases.

Example Scenarios

Scenario 1: Quick Test Migration

This scenario tests the migration process without committing to a full migration. Run option 2 (Pre-Migration) to validate the environment, run option 9 (Print Summary SOURCE) to review source databases, run option 3 (Dump Databases) to create a backup, run option 10 (Print Summary DEST) to verify the destination is ready, and stop here to review without making changes to the destination. This approach is ideal for initial assessment before production migration.

Scenario 2: Staged Migration with Validation

This scenario migrates databases with extensive validation between steps. Run option 2 (Pre-Migration), option 3 (Dump Databases), option 4 (Compress and Checksum), option 5 (Transfer to Destination), option 6 (Restore Databases), option 8 (Data Validation Suite) to verify data integrity, option 10 (Print Summary DEST) to compare database sizes, option 7 (Post-Restore Maintenance) to optimize databases, option 11 (Backup & Restore Server Files), option 13 (Setup bi_cube) if applicable, options 14-17 for configuration sync, and option 19 (Final Cleanup). This approach is ideal for production migration with checkpoints for validation.

Scenario 3: Configuration-Only Migration

This scenario migrates only server configuration files without touching databases. Run option 2 (Pre-Migration), option 4 (Compress and Checksum) to archive server files, option

5 (Transfer to Destination), extract manually or run option 6 but skip database restore, option 11 (Backup & Restore Server Files), option 12 (Rename Smoothie Folder), option 13 (Setup bi_cube), options 14-17 for system configuration, and option 19 (Final Cleanup). This approach is ideal for updating server configuration without database migration.

Scenario 4: Database-Only Migration

This scenario migrates only databases, skipping server file configuration. Run option 2 (Pre-Migration), option 3 (Dump Databases), option 4 (Compress and Checksum), option 5 (Transfer to Destination), option 6 (Restore Databases), option 7 (Post-Restore Maintenance), option 8 (Data Validation Suite), and option 19 (Final Cleanup). This approach is ideal for database refresh on an existing configured server.

Scenario 5: Recovery from Failed Migration

This scenario resumes migration after a failure at the transfer stage. Verify archives exist on source using ls -lh /tmp/pg_dumps.tar.zst, run option 5 (Transfer to Destination) to retry transfer, option 6 (Restore Databases), option 7 (Post-Restore Maintenance), option 8 (Data Validation Suite), option 11 (Backup & Restore Server Files), options 13-17 for configuration, and option 19 (Final Cleanup). This approach is ideal for recovering from network interruption during transfer.

Scenario 6: Incremental Testing

This scenario tests each phase independently before full migration. On Day 1, run option 2 (Pre-Migration), option 3 (Dump Databases), verify dumps by SSH to source and checking /tmp/pg_migration/dumps/, and run option 19 (Cleanup) to remove test dumps. On Day 2, run option 3 (Dump Databases), option 4 (Compress and Checksum), option 5 (Transfer to Destination), verify on destination using ls -lh /tmp/pg_dumps.tar.zst, and run option 19 (Cleanup). On Day 3, complete steps from Day 2, run option 6 (Restore Databases), option 10 (Print Summary DEST), and option 8 (Data Validation Suite). On Day 4, run option 1 (Full Migration). This approach is ideal for risk-averse migration with thorough testing.

Scenario 7: bi_cube Environment Setup Only

This scenario configures bi_cube on an already-migrated server. Run option 18 (Check bi_cube Detection) to verify detection, and if detected, run option 13 (Setup bi_cube). Verify the Python environment by SSH to destination and running source /opt/bi_cube_ip_whitelist/bin/activate followed by pip list. This approach is ideal for post-

migration bi_cube configuration.

Scenario 8: Hostname and Configuration Sync

This scenario synchronizes system configuration without database migration. Run option 14 (Sync Timezone), option 15 (Update /etc/hosts), option 16 (Update .bashrc PS1 Prompt), and option 17 (Apply Mambo Cron Schedules). SSH to destination and verify using timedatectl, cat /etc/hosts, cat ~/.bashrc | grep PS1, and crontab -l. This approach is ideal for configuration drift correction on existing servers.

Scenario 9: Parallel Multi-Server Migration

This scenario migrates multiple servers simultaneously using different jumpbox sessions. In Terminal 1, run SOURCE_HOST=server1 DEST_HOST=newserver1 ./pg-migrator.sh and select option 1 (Full Migration). In Terminal 2, run SOURCE_HOST=server2 DEST_HOST=newserver2 ./pg-migrator.sh and select option 1. In Terminal 3, run SOURCE_HOST=server3 DEST_HOST=newserver3 ./pg-migrator.sh and select option 1. This approach is ideal for large-scale infrastructure migration.

Scenario 10: Maintenance-Only Run

This scenario runs database maintenance on destination without migration. Set DEST_HOST only (no source needed), run option 7 (Post-Restore Maintenance), monitor execution time for each operation, and run option 10 (Print Summary DEST) to verify optimization. This approach is ideal for regular database maintenance or post-migration optimization.

Best Practices

Always run Pre-Migration first to validate the environment and identify potential issues before starting the migration. Monitor disk space throughout the process to avoid running out of space during critical operations. Keep the source server running until validation is complete to allow for rollback if needed. Document custom configurations before migration to ensure nothing is lost during the process. Test on non-production servers first to identify any issues specific to your environment. Schedule during maintenance windows to minimize impact on users. Verify bi_cube detection before full migration if applicable to ensure proper configuration. Keep backups of critical data independent of migration as an additional safety measure. Review execution times to plan production windows accurately.

Use a staged approach for critical production systems to minimize risk.

Post-Migration Checklist

After completing the migration, verify all databases were restored using option 10, validate row counts match the source, check application connectivity to ensure services are working, verify cron jobs are running as expected, test bi_cube functionality if applicable, confirm timezone is correct, verify hostname resolution, test DW start/stop commands, review PostgreSQL logs for errors, update monitoring systems with new server information, update documentation to reflect the migration, and schedule source server decommission after a successful validation period.

Support and Maintenance

Log Locations

PostgreSQL logs are located at /var/log/postgresql/, system logs at /var/log/syslog, and script output is displayed on the console (redirect to a file if needed for later review).

Common Issues

If you encounter “Cannot connect to source/destination via SSH”, verify SSH keys are properly configured, check firewall rules allow SSH connections, and test manual SSH connection to identify the issue.

If you see “Insufficient disk space”, clean up temporary files, increase volume size, or use a different backup location with more available space.

If “Database dump failed” occurs, check PostgreSQL is running on the source server, verify user permissions for the postgres user, and review PostgreSQL logs for specific error messages.

If “bi_cube setup failed” appears, verify Python 3 is installed on the destination server, check internet connectivity for pip package installation, and review package dependencies for conflicts.

Security Considerations

Use dedicated SSH keys for migration and rotate them after completion to maintain security. The script requires sudo access, so review commands before execution to understand what will be executed. Data is transferred via SSH which provides encryption in transit. Temporary files are cleaned up automatically, but verify with option 19 to ensure no sensitive data remains. No credentials are stored in the script as it uses PostgreSQL peer authentication for database access.

Performance Metrics

Typical execution times vary by database size and hardware configuration. Pre-Migration takes 1-2 minutes, Database Dump takes 10-60 minutes depending on size, Compression takes 5-15 minutes, Transfer takes 10-30 minutes depending on network speed, Restore takes 15-90 minutes depending on size, Maintenance takes 20-120 minutes depending on size, and Configuration takes 5-10 minutes. The total migration time typically ranges from 1-6 hours for a complete migration.

Version History

Version 1.0 was released on 15 October 2025 as the initial release. Version 1.1 was released on 31 October 2025 with added rsync utilities and improved error handling.

Author

Frank Claassens
Created: 15 October 2025
Updated: 31 October 2025

License