Introduction

This document explains the details of the Backup System. It includes how the command-line interface (CLI) works, the available commands, how scripts are used. The design follows the structure of our system diagrams, including UML, component, and data models.

Command-Line Interface (CLI) Specification

The CLI allows administrators to manage the Backup System using commands. It supports command-line arguments for flexibility and provides features like scheduling backups, running them manually, restoring data, and configuring settings.

Available Commands

Command	*Description*	*Syntax*
help	Displays the list of available commands.	backup-system help
schedule_backup	Configures a new backup schedule.	backup-system schedule_backup --target <TARGET_NAME> --type <TYPE> --schedule "<CRON_EXPRESSION>"
run_backup	Manually triggers a backup for a specific target.	backup-system run_backup --target <TARGET_NAME>
restore_backup	Restores data from a backup.	backup-system restore_backup --id <BACKUP_ID>
status	Displays the current status of the backup system.	backup-system status
list_backups	Lists all stored backups with metadata.	backup-system list_backups
enable_server	Enables a server for backups.	backup-system enable_server --target <TARGET_NAME>
disable_server	Disables a server, stopping backups for it.	backup-system disable_server --target <TARGET_NAME>
validate_script	Validates pre- and post-backup scripts to ensure they are functional.	backup-system validate_script --path <SCRIPT_PATH>
exit	Exits the program.	exit

Detailed Command Syntax and Examples

1. schedule_backup

- **Purpose**: Configure a new schedule for backups.
- **Syntax**:
- backup-system schedule_backup --target <TARGET_NAME> --type <TYPE> --schedule "<CRON_EXPRESSION>"
- **Arguments**:
    - --target: Name of the backup target (e.g., Server1, Database1).
    - --type: Type of backup (full or incremental).
    - --schedule: Cron-like schedule (e.g., "0 2 * * 0" for every Sunday at 2 AM).
- **Example**:
- backup-system schedule_backup --target Server1 --type full --schedule "0 3 * * 1"
    - This schedules a **full backup** for Server1 every Monday at 3 AM.

2. run_backup

- **Purpose**: Manually triggers a backup for a specific target.
- **Syntax**:
- backup-system run_backup --target <TARGET_NAME>
- **Arguments**:
    - --target: Name of the target to back up (e.g., Server1 or Database1).
- **Example**:
- backup-system run_backup --target Database1
    - This runs a backup for Database1 immediately.

3. restore_backup

- **Purpose**: Restore data from a specific backup.
- **Syntax**:
- backup-system restore_backup --id <BACKUP_ID>
- **Arguments**:
    - --id: ID of the backup to restore.

- **Example**:
- backup-system restore_backup --id 12345
  - This restores the backup with ID 12345.

5. validate_script

- **Purpose**: Ensures the provided script works before assigning it.
- **Syntax**:
- backup-system validate_script --path <SCRIPT_PATH>
- **Arguments**:
  - --path: Path to the script to validate.
- **Example**:
- backup-system validate_script --path /scripts/pre_backup.sh
- **Output**:
  - Success: "Script validated successfully."
  - Failure: "Script validation failed: Syntax error at line 10."

Integration of Scripts

Purpose

Scripts allow administrators to perform specific tasks before and after backups, such as stopping services or cleaning up temporary files.

Configuration

Scripts are defined in the configuration file, like this:

```
servers:
  - name: Server1
    pre_backup_script: /scripts/pre_backup_server1.sh
    post_backup_script: /scripts/post_backup_server1.sh
databases:
  - name: Database1
    pre_backup_script: /scripts/pre_backup_db1.sh
    post_backup_script: /scripts/post_backup_db1.sh
```