Database Backup & Restore System

🎯 Overview

A complete backup and restore system with 5 scripts:

Script	Purpose	Use Case
`backup.sh`	Create timestamped backup	Regular backups, before changes
`restore.sh`	Restore from any backup	Recover from issues, migrate data
`rollback.sh`	Quick restore to latest	Emergency rollback
`list-backups.sh`	View all backups	Check available backups
`cleanup-backups.sh`	Remove old backups	Save disk space

🚀 Quick Start

```bash
# Make all scripts executable
chmod +x backup.sh restore.sh rollback.sh list-backups.sh cleanup-backups.sh

# Create your first backup
./backup.sh

# View all backups
./list-backups.sh

# Restore if needed
./restore.sh
```

📝 Detailed Usage

1. Creating Backups

Create a backup:

```bash
./backup.sh
```

What it does:

- Creates timestamped backup: `backup_20241205_143022.sql.gz`
- Compresses with gzip (saves ~90% space)
- Keeps last 10 backups (configurable)
- Auto-deletes old backups

Example output:

```
════════════════════════════════════════
  Database Backup
════════════════════════════════════════

✓ PostgreSQL is running
➜ Creating backup...
  Target: ./backups/backup_20241205_143022.sql

✓ Backup created successfully
  File: ./backups/backup_20241205_143022.sql
  Size: 2.3M

✓ Backup compressed
  File: ./backups/backup_20241205_143022.sql.gz
  Size: 234K

✓ All backups within limit
```

When to backup:

- Before major changes
- Before database migrations
- Daily (can automate with cron)
- Before updates/deployments

2. Viewing Backups

List all backups:

```bash
./list-backups.sh
```

Example output:

```
════════════════════════════════════
  Database Backups
════════════════════════════════════

Total backups: 5
  Regular backups: 4
  Safety backups: 1

Total size: 1.2M

── Regular Backups ──

FILENAME                          SIZE      DATE
───────────────────────────────────────────────────────────────────
backup_20241205_143022.sql.gz     234K      20241205 14:30:22
backup_20241205_120000.sql.gz     228K      20241205 12:00:00
backup_20241204_180000.sql.gz     198K      20241204 18:00:00
backup_20241204_120000.sql.gz     201K      20241204 12:00:00

── Safety Backups (Pre-Restore) ──

FILENAME                          SIZE      DATE
───────────────────────────────────────────────────────────────────
pre_restore_20241205_141530.sql.gz 232K      20241205 14:15:30
```

3. Restoring from Backup

Interactive restore (choose from list):

```bash
./restore.sh
```

Direct restore (specify file):

```bash
./restore.sh ./backups/backup_20241205_143022.sql.gz
```

What happens:

1. Shows selected backup info
2. Warns about data loss
3. Creates safety backup of current data
4. Stops backend to prevent connections
5. Drops and recreates database
6. Restores backup data
7. Restarts backend
8. Verifies restore

Example output:

```
────────────────────────────────────────
 Database Restore
────────────────────────────────────────

✓ PostgreSQL is running

✓ Selected backup: backup_20241205_143022.sql.gz
  Size: 234K
  Date: 20241205 14:30:22

⚠️  WARNING: This will OVERWRITE the current database!

Current database contents will be LOST.
All users, projects, chats, and messages will be replaced with backup data.

Are you sure you want to continue? [y/N]: y

→ Creating safety backup of current database...
✓ Safety backup created: ./backups/pre_restore_20241205_151030.sql.gz

→ Stopping backend service...
✓ Backend stopped

→ Decompressing backup...
✓ Backup decompressed

→ Dropping and recreating database...
✓ Database recreated

→ Restoring backup...
✓ Backup restored successfully

→ Starting backend service...
✓ Backend started

→ Verifying restore...

Database statistics:
  Users: 5
  Projects: 12
  Chat sessions: 34

✓ Restore complete!
```

4. Quick Rollback

Emergency rollback to latest backup:

```bash
./rollback.sh
```

Use when:

- Something went wrong after a change
- Need to quickly undo recent actions
- Emergency recovery needed

What it does:

- Automatically selects most recent backup
- Runs restore process
- No need to choose from list

Example output:

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
  Quick Database Rollback
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Most recent backup:
  File: backup_20241205_143022.sql.gz
  Size: 234K
  Date: 20241205 14:30:22

⚠️  This will rollback to the above backup!

Continue with rollback? [y/N]: y

[... restore process runs ...]

✓ Rollback completed successfully!
```

5. Cleaning Old Backups

Clean old backups:

```bash
./cleanup-backups.sh
```

What it does:

- Shows current backup counts
- Asks how many to keep
- Shows files to be deleted
- Confirms before deletion
- Can also clean safety backups

Example output:

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
  Backup Cleanup
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Current backup counts:
  Regular backups: 15
  Safety backups: 3

How many regular backups would you like to keep?
Keep last N backups [default: 10]: 5

⚠ Will remove 10 old regular backup(s)

Files to be deleted:
  backup_20241201_120000.sql.gz (198K)
  backup_20241202_120000.sql.gz (201K)
  [... 8 more files ...]

Delete these backups? [y/N]: y
✓ Regular backups cleaned

⚠ Found 3 safety backup(s) (pre-restore backups)

Safety backups are created before each restore operation.
They can be safely deleted once you're sure the restore was successful.

Delete all safety backups? [y/N]: y
✓ Safety backups deleted (freed ~700K)

✓ Cleanup complete

Remaining backups:
  Regular: 5
  Safety: 0
```

⚙ Configuration

Backup Retention

Edit `backup.sh` to change retention:

```bash
KEEP_BACKUPS=10  # Change this number
```

Backup Location

Change backup directory in all scripts:

```bash
BACKUP_DIR="./backups"  # Change to your preferred location
```

🔄 Automation

Daily Backups with Cron

Add to crontab (`crontab -e`):

```bash
# Daily backup at 2 AM
0 2 * * * cd /path/to/ai-platform && ./backup.sh >> /var/log/ai-platform-backup.log 2>&1

# Weekly cleanup (keep last 30)
0 3 * * 0 cd /path/to/ai-platform && echo "30" | ./cleanup-backups.sh >> /var/log/ai-platform-cleanup.log 2>&1
```

Pre-Deployment Backup

Add to your deployment script:

```bash
#!/bin/bash
echo "Creating pre-deployment backup..."
./backup.sh

echo "Deploying changes..."
# ... your deployment commands ...

if [ $? -ne 0 ]; then
    echo "Deployment failed! Rolling back..."
    ./rollback.sh
fi
```

🔐 Security Considerations

Backup File Security

Backups contain sensitive data!

```bash
# Restrict access
chmod 700 backups/
chmod 600 backups/*.gz

# Encrypt backups (optional)
gpg --symmetric --cipher-algo AES256 backup_20241205_143022.sql.gz

# Decrypt when needed
gpg --decrypt backup_20241205_143022.sql.gz.gpg | gunzip | docker-compose exec -T postgres psql -U aiplatform -d aiplatfo
```

Remote Backup Storage

Upload to cloud storage:

```bash
# AWS S3
aws s3 cp ./backups/backup_*.sql.gz s3://my-bucket/backups/

# rsync to remote server
rsync -av --delete ./backups/ user@backup-server:/backups/ai-platform/
```

📊 Backup Strategies

Development Environment

- Backup before major changes
- Keep 5-10 recent backups
- Clean old backups regularly

Production Environment

- **Daily backups** (automated)
- Keep last 30 daily backups
- Keep weekly backups for 3 months
- Keep monthly backups for 1 year
- Store backups off-site
- Test restore regularly

Recommended Schedule

```bash
# Daily at 2 AM
0 2 * * * ./backup.sh

# After each deployment
# (manual or in CI/CD pipeline)

# Before database migrations
# (manual)
```

🧪 Testing Backups

Always test your backups!

```bash
# 1. Create test database
docker-compose exec postgres psql -U aiplatform -d postgres -c "CREATE DATABASE test_restore;"

# 2. Restore to test database
gunzip -c ./backups/backup_20241205_143022.sql.gz | \
  docker-compose exec -T postgres psql -U aiplatform -d test_restore

# 3. Verify data
docker-compose exec postgres psql -U aiplatform -d test_restore -c "SELECT COUNT(*) FROM users;"

# 4. Clean up
docker-compose exec postgres psql -U aiplatform -d postgres -c "DROP DATABASE test_restore;"
```

🚨 Troubleshooting

"No space left on device"

```bash
# Check backup size
du -sh ./backups

# Clean old backups
./cleanup-backups.sh

# Move to larger disk
mv ./backups /mnt/storage/backups
ln -s /mnt/storage/backups ./backups
```

"Database is in use"

```bash
# Force disconnect all users
docker-compose stop backend

# Then restore
./restore.sh
```

"Backup file corrupted"

```bash
# Test gzip integrity
gunzip -t ./backups/backup_20241205_143022.sql.gz

# If corrupted, use older backup
./restore.sh ./backups/backup_20241204_120000.sql.gz
```

"Restore taking too long"

Large databases may take time:

- 1 GB: ~2-5 minutes
- 10 GB: ~20-30 minutes
- 100 GB: ~3-5 hours

Monitor progress:

```bash
# Watch PostgreSQL logs
docker-compose logs -f postgres
```

📈 Best Practices

1. **Backup before changes** - Always backup before major operations
2. **Test restores regularly** - Verify backups work
3. **Multiple backup locations** - On-site + off-site
4. **Automate backups** - Use cron for consistency
5. **Monitor backup size** - Watch for unexpected growth
6. **Document restore procedures** - Team should know how
7. **Version backups** - Keep multiple versions
8. **Encrypt sensitive data** - Use GPG for production
9. **Clean old backups** - Don't fill disk
10. **Alert on failures** - Monitor backup success

🔗 Related Commands

```bash
# Manual PostgreSQL backup (without scripts)
docker-compose exec -T postgres pg_dump -U aiplatform aiplatform > manual_backup.sql

# Manual restore (without scripts)
docker-compose exec -T postgres psql -U aiplatform -d aiplatform < manual_backup.sql

# Backup specific tables only
docker-compose exec -T postgres pg_dump -U aiplatform -t users -t projects aiplatform > partial_backup.sql

# Backup with data only (no schema)
docker-compose exec -T postgres pg_dump -U aiplatform --data-only aiplatform > data_only.sql

# Backup with schema only (no data)
docker-compose exec -T postgres pg_dump -U aiplatform --schema-only aiplatform > schema_only.sql
```