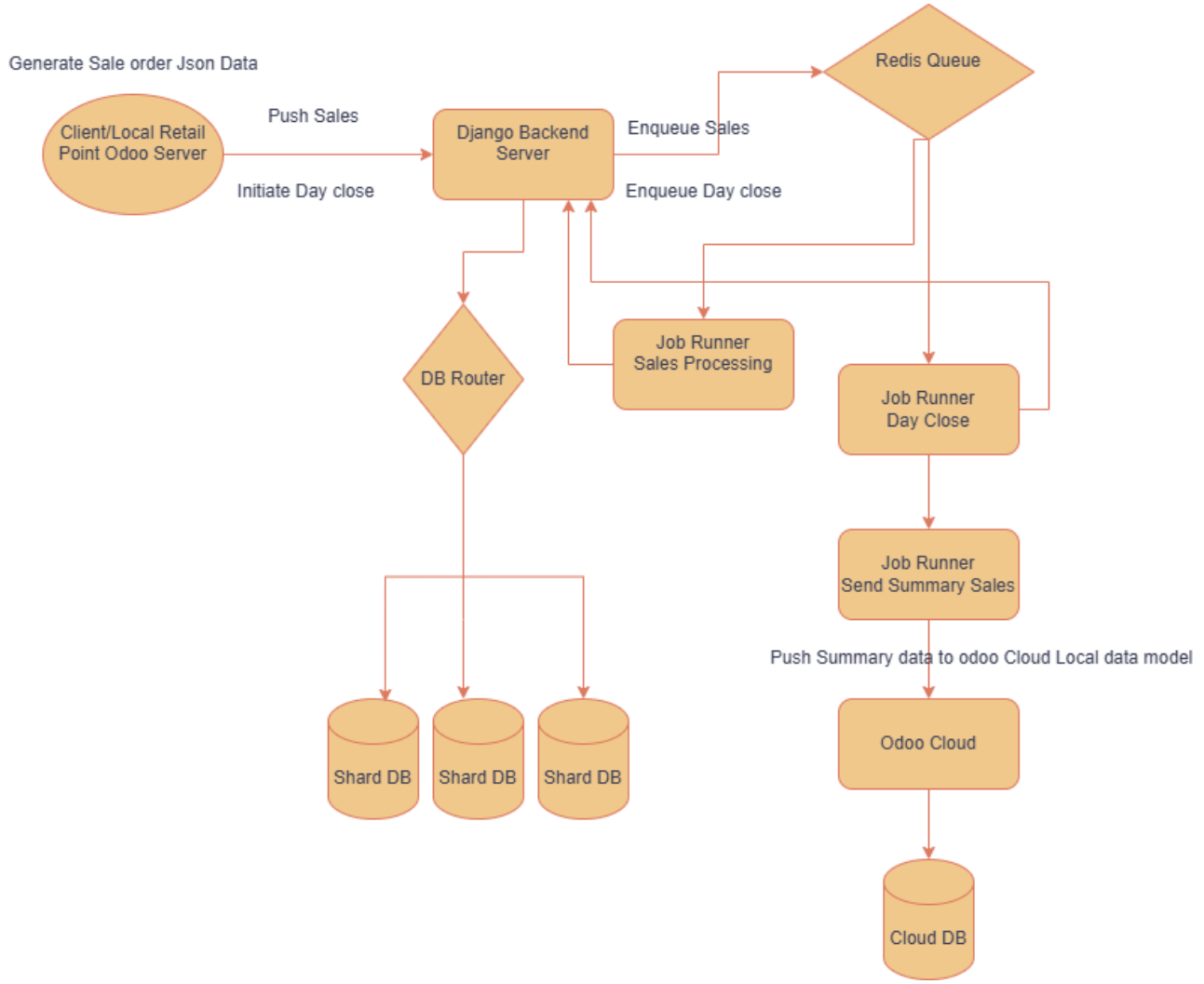
CFED FLS

Distributed FLS Sales Solution Documentation

Architecture Overview

This solution is designed to process retail sales data locally and synchronize it with an Odoo Cloud instance for centralized reporting and management. The architecture ensures high availability, scalability, and efficient handling of tasks, including sale processing and day-close operations.

Components and Workflow



1. JSON Data Generation

- Client/Local Retail Point Odoo Server:
  - Generates sale order data in JSON format.

- Initiates sales and day-close tasks for processing.

2. Django Backend Server

- Acts as the central processing unit for incoming tasks.
- Responsibilities:
  - Accepts sale and day-close requests from clients.
  - Routes database operations using the DB Router.
  - Enqueues tasks into Redis for asynchronous processing by RQ workers.

3. Redis Queue

- A high-performance message broker used to store and manage tasks for asynchronous processing.
- Handles the following:
  - Sales Tasks: Queues individual sales for processing.
  - Day-Close Tasks: Queues day-end operations for summary preparation.

4. Job Runners (RQ Workers)

- Workers poll Redis queues for tasks and process them asynchronously.
- Types of Job Runners:
  - Sales Processing Worker:
    - Processes sale orders.
    - Updates shard databases.
  - Day-Close Worker:
    - Handles day-end operations.
    - Prepares sales summary data.
  - Send Summary Worker:
    - Pushes summarized data to the Odoo Cloud.

5. Database Management

- DB Router:
  - Ensures correct routing of database operations across sharded databases.
  - Shard DBs store localized data for performance optimization.

6. Odoo Cloud Integration

- Summarized data from the day-close operation is sent to the Odoo Cloud.
- Responsibilities:
  - Centralized storage and management of retail data.

- Data is stored in a Cloud Database for global reporting.

**Data Flow

- Client Application(JSON Data) → Django REST Endpoint → RQ Worker → Sharded DB.
- **Daily Close(initiated by Client) → Summary Aggregation(Django) → Post to Odoo Central Cloud local data → Generate Summary Sales

---

Technologies Used

1. Backend Framework

- Django:
  - Provides the backend framework for managing the business logic, REST APIs, and database routing.
- Django RQ:
  - Manages task queues using Redis.
  - Simplifies the integration of RQ workers into Django.

2. Task Queue

- Redis:
  - Serves as the message broker for queueing tasks.
  - Used with Django RQ for task management.

3. Workers

- RQ Workers:
  - Asynchronous task processing.
  - Processes sales and day-close tasks efficiently in distributed environments.

4. Database

- Sharded Databases:
  - Localized storage for high performance and fault isolation.
- Timescale Database Extension:
  - Hyper tables for the better query result for sale order and lot details per date filter, the data will be day interval chunks.

Deployment Steps

1. Configure Django RQ

- Add the following to `settings.py`:

```
RQ_QUEUES = {
    'default': {
        'HOST': 'localhost',
        'PORT': 6379,
        'DB': 0,
        'DEFAULT_TIMEOUT': 360,
    },
    'sales_processing': {
        'HOST': 'localhost',
        'PORT': 6379,
        'DB': 1,
        'DEFAULT_TIMEOUT': 360,
    },
    'day_close': {
        'HOST': 'localhost',
        'PORT': 6379,
        'DB': 2,
        'DEFAULT_TIMEOUT': 360,
    },
     'send_summary': {
        'HOST': 'localhost',
        'PORT': 6379,
        'DB': 3,
        'DEFAULT_TIMEOUT': 360,
    },
}
```

**2.## Deployment Setup

Step-by-Step Setup Instructions

1. Preparation

Make sure you have the required permissions and the necessary software packages installed.

2. Bash Script for Deployment Setup

```
#!/bin/bash

# Function to check if command exists
command_exists() {
    command -v "$1" >/dev/null 2>&1
```

```
}

# Variables
USER_HOME=$(eval echo ~$USER) # Get the home directory of the current user
PROJECT_DIR="$(dirname \"$(realpath \"$0\")\")" # The Django project path in
the user's home directory
VENV_DIR="$PROJECT_DIR/venv" # Virtual environment directory inside the
project
LOG_DIR="/var/log/supervisor"
SUPERVISOR_CONF="/etc/supervisor/conf.d/django_project.conf"
DB_USER="django"
DB_PASSWORD="secure_password"

# Add PostgreSQL repository
echo "Adding PostgreSQL repository..."
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release
-cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-
key add -

# Add TimescaleDB repository
echo "Adding TimescaleDB repository..."
echo "deb https://packagecloud.io/timescale/timescaledb/ubuntu/
$(lsb_release -c -s) main" | sudo tee
/etc/apt/sources.list.d/timescaledb.list
wget --quiet -O - https://packagecloud.io/timescale/timescaledb/gpgkey |
sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/timescaledb.gpg

# Update package list again
sudo apt-get update

sudo apt-get update -y
sudo apt-get install -y apt-transport-https
sudo apt-get install -y supervisor python3-venv python3-pip wget gnupg lsb-
release
sudo apt-get install -y postgresql-14
sudo apt-get install -y redis-server

# Step 2: Enable and Start Redis Server
echo "Configuring Redis server..."
sudo systemctl enable redis
sudo systemctl start redis

# Step 2: Install TimescaleDB
echo "Installing TimescaleDB..."
sudo apt-get install -y timescaledb-2-postgresql-14

# Configure TimescaleDB
echo "Configuring TimescaleDB..."
sudo timescaledb-tune --quiet --yes
```

```
sudo systemctl restart postgresql

# Step 3: Configure PostgreSQL
echo "Configuring PostgreSQL..."
sudo -u postgres psql -c "CREATE USER $DB_USER WITH PASSWORD
'$DB_PASSWORD';"
sudo -u postgres psql -c "ALTER USER $DB_USER CREATEDB;"

# Step 4: Create log directory
echo "Creating log directory..."
sudo mkdir -p $LOG_DIR
sudo chown $USER:$USER $LOG_DIR

# Step 5: Set up Virtual Environment
if [ ! -d "$VENV_DIR" ]; then
  echo "Creating virtual environment in $VENV_DIR..."
  python3 -m venv $VENV_DIR
fi

# Activate Virtual Environment
echo "Activating virtual environment..."
source $VENV_DIR/bin/activate

# Step 6: Install Python requirements
if [ -f "$PROJECT_DIR/requirements.txt" ]; then
  echo "Installing Python requirements..."
  pip install --upgrade pip
  pip install -r $PROJECT_DIR/requirements.txt
else
  echo "requirements.txt not found in $PROJECT_DIR. Skipping installation."
fi

# Step 7: Django initial setup - Migrate Database and Custom Commands
echo "Running Django management commands..."
python3 $PROJECT_DIR/manage.py create_shard_database
python3 $PROJECT_DIR/manage.py enable_timescale
python3 $PROJECT_DIR/manage.py migrate
python3 $PROJECT_DIR/manage.py migrate_shards

# Step 8: Create Supervisor configuration file
echo "Creating Supervisor configuration..."
sudo tee $SUPERVISOR_CONF > /dev/null <<EOL
[supervisord]
nodaemon=true

[program:django]
command=$VENV_DIR/bin/python $PROJECT_DIR/manage.py runserver 0.0.0.0:5000
directory=$PROJECT_DIR
autostart=true
autorestart=true
```

```
stdout_logfile=$LOG_DIR/django.log
stderr_logfile=$LOG_DIR/django_error.log

[program:send_summary]
command=$VENV_DIR/bin/python3 $PROJECT_DIR/manage.py rqworker send_summary
directory=$PROJECT_DIR
numprocs=2
process_name=%(program_name)s_%(process_num)02d
autostart=true
autorestart=true
stdout_logfile=$LOG_DIR/rq_send_summary.log
stderr_logfile=$LOG_DIR/rq_send_summary_error.log

[program:rq_sales_processing]
command=$VENV_DIR/bin/python3 $PROJECT_DIR/manage.py rqworker
sales_processing
directory=$PROJECT_DIR
numprocs=12
process_name=%(program_name)s_%(process_num)02d
autostart=true
autorestart=true
stdout_logfile=$LOG_DIR/rq_sales_processing.log
stderr_logfile=$LOG_DIR/rq_sales_processing_error.log

[program:rq_day_close]
command=$VENV_DIR/bin/python3 $PROJECT_DIR/manage.py rqworker day_close
directory=$PROJECT_DIR
numprocs=8
process_name=%(program_name)s_%(process_num)02d
autostart=true
autorestart=true
stdout_logfile=$LOG_DIR/rq_day_close_%(process_num)02d.log
stderr_logfile=$LOG_DIR/rq_day_close_error_%(process_num)02d.log
EOL

# Step 9: Update Supervisor configuration
echo "Updating Supervisor service..."
sudo supervisorctl reread
sudo supervisorctl update

# Step 10: Start Supervisor services
echo "Starting Supervisor services..."
sudo systemctl enable supervisor
sudo systemctl start supervisor

# Status check
echo "Supervisor Status:"
sudo supervisorctl status

echo "Setup Complete!"
```

```
echo "User: $DB_USER"
echo "Password: $DB_PASSWORD"
```

Notes

- Ensure that `requirements.txt` contains all necessary dependencies.
- Modify the script variables (e.g., `DB_USER` and `DB_PASSWORD`) as per your project requirements.
- Use this script as a reference for deploying your Django project with PostgreSQL, TimescaleDB,

Integrate Odoo Cloud

- Use REST API for creating Local data in cfed_sales_customisation/controller/local_data_api.py:

  ```
  ```

  @http.route('/api/sales-summary', type='json', auth="public",methods=['POST'])
  def create_sales_summary_local_data(self, **post):
  try:
  raw_json_data = request.httprequest.data
  data = json.loads(raw_json_data)
  local_data = {
  'retail_point_id': data.get('retail_point_id'),
  'data': data,
  'push_type': 'sales',
  'date': data.get('sale_date'),
  'local_record_id': False,
  'is_latest_data': True,
  'total_including_miscellaneous': data.get('total_including_miscellaneous',0),
  'liquor_amount_total':data.get('amount_total'),
  'payment_journal_id': data.get('payment_journal_id'),
  'sale_num': data.get('sale_num'),
  }
  history= request.env['cfed.local.data'].sudo().create(local_data)
  return {"local_data_id":history.id,"status":"Success"}

```
except Exception as e:
    return {'status': 'Failed', 'message': str(e)}
```