Django REST API, Docker and Snowflake Container

Step 1: Create a Python Virtual Environment

```
python -m venv env
./env/Scripts/activate
```

Step 2: Install Required Packages

```
pip install django djangorestframework django-cors-headers django-snowflake==
5.2.*
```

Step 3: Create a Django Project

```
django-admin startproject backend
cd backend
```

Step 4: Create a Django App

```
python manage.py startapp api
```

Step 5: Configure Settings

Edit `backend/settings.py`:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'corsheaders',
    'api',
]

MIDDLEWARE = [
    ...
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]

# Add at the bottom of settings.py
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",  # React's default port
]

REST_FRAMEWORK = {
```

Django REST API, Docker and Snowflake Container

```
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

Step 6: Create Models

Edit `api/models.py`:

```
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

Step 7: Create Serializers

Create `api/serializers.py`:

```
from rest_framework import serializers
from .models import Task

class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Task
        fields = '__all__'
```

Step 8: Create Views

Edit `api/views.py`:

```
from rest_framework import viewsets
from .models import Task
from .serializers import TaskSerializer

class TaskViewSet(viewsets.ModelViewSet):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
```

Django REST API, Docker and Snowflake Container

Step 9: Configure URLs

Create `api/urls.py`:

```
from django.urls import path, include
from rest_framework import routers
from . import views

router = routers.DefaultRouter()
router.register(r'tasks', views.TaskViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

Edit `backend/urls.py`:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]
```

Step 10: Run Migrations

```
python manage.py makemigrations
python manage.py migrate
```

Step 11: Create Superuser

```
python manage.py createsuperuser
```

Step 12: Run the Server

```
python manage.py runserver
```

Your API is now running at http://localhost:8000/api/tasks/

Django REST API, Docker and Snowflake Container

Creating a Docker Image for Django REST API

1. Create requirements.txt file

First, let's create a `requirements.txt` file with all the required Python packages:

```
pip freeze > requirements.txt
```

```
asgiref==3.8.1
asn1crypto==1.5.1
boto3==1.38.36
botocore==1.38.36
certifi==2025.6.15
cffi==1.17.1
charset-normalizer==3.4.2
cryptography==45.0.4
Django==5.2.3
django-cors-headers==4.7.0
django-snowflake==5.2
djangorestframework==3.16.0
filelock==3.18.0
idna==3.10
jmespath==1.0.1
packaging==25.0
platformdirs==4.3.8
pycparser==2.22
PyJWT==2.10.1
pyOpenSSL==25.1.0
python-dateutil==2.9.0.post0
pytz==2025.2
requests==2.32.4
s3transfer==0.13.0
six==1.17.0
snowflake-connector-python==3.15.0
sortedcontainers==2.4.0
sqlparse==0.5.3
tomlkit==0.13.3
typing_extensions==4.14.0
tzdata==2025.2
urllib3==2.4.0
```

2. Create Dockerfile

Django REST API, Docker and Snowflake Container

Create a `Dockerfile` in your project root with the following content:

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Set work directory
WORKDIR /code

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements file
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy project
COPY . .

# Expose the port the app runs on
EXPOSE 8000

# Command to run the application
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

3. Create docker-compose.yml file

Create a `docker-compose.yml` file for easier container management:

```
services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
```

Django REST API, Docker and Snowflake Container

```
      - .:/code
    ports:
      - "8000:8000"
```

5. Build and run the Docker containers

Run these commands in sequence:

```
# Build the containers
docker-compose build

# Run the containers
docker-compose up

# In a new terminal, run migrations
docker-compose exec web python manage.py migrate

# Create superuser
docker-compose exec web python manage.py createsuperuser
```

6. Access the application

- API will be available at: http://localhost:8000/api/tasks/
- Django admin at: http://localhost:8000/admin/

Additional Notes

1. For development, you might want to add a `.dockerignore` file to exclude unnecessary files:

text

Copy

Download

```
.env
*.pyc
*.pyo
*.pyd
__pycache__
```

https://www.linkedin.com/in/sadrulalom/

```
*.sqlite3
*.db
*.log
*.pot
*.py[co]
*.egg-info
*.so
.DS_Store
.git
.gitignore
.venv
env/
venv/
```

Snowflake Container Service Setup for Django REST API

1. Setup SQL File (`setup.sql`)

This file creates the necessary Snowflake objects (database, warehouse, role, etc.) for your container service.

```sql
-- Create database and schema
CREATE DATABASE IF NOT EXISTS DJANGO_API;
CREATE SCHEMA IF NOT EXISTS DJANGO_API.CONTAINERS;

-- Create warehouse for container services
CREATE WAREHOUSE IF NOT EXISTS DJANGO_WH
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 300
  AUTO_RESUME = TRUE;

-- Create role and grant permissions
CREATE ROLE IF NOT EXISTS DJANGO_ADMIN;
GRANT USAGE ON WAREHOUSE DJANGO_WH TO ROLE DJANGO_ADMIN;
GRANT USAGE ON DATABASE DJANGO_API TO ROLE DJANGO_ADMIN;
GRANT ALL ON SCHEMA DJANGO_API.CONTAINERS TO ROLE DJANGO_ADMIN;
```

```
-- Grant role to your user
GRANT ROLE DJANGO_ADMIN TO USER <your_username>;
```

2. Manifest File (`manifest.yml`)

This defines the container service specification and configuration.

```yaml
spec_version: 1
containers:
- name: django-api
  image: /DJANGO_API/CONTAINERS/IMAGES/DJANGO_API_IMAGE
  env:
    DATABASE_NAME: DJANGO_API
    DATABASE_SCHEMA: CONTAINERS
    DJANGO_SECRET_KEY: your-secret-key-here
    ALLOWED_HOSTS: "*"
    CORS_ALLOWED_ORIGINS: "*"
  resources:
    cpu: 1
    memory: 2Gi
  endpoints:
  - name: django-api-endpoint
    port: 8000
    public: true
```

3. Backend Configuration (`backend.yaml`)

This Kubernetes-style configuration file defines the backend service for Snowflake.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: django-backend
spec:
  selector:
    app: django-api
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
```

```
  type: ClusterIP
```

4. Dockerfile Modifications

Your existing Dockerfile needs a few adjustments for Snowflake:

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
ENV DJANGO_SETTINGS_MODULE=backend.settings

# Set work directory
WORKDIR /code

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements file
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy project
COPY . .

# Collect static files
RUN python manage.py collectstatic --noinput

# Expose the port the app runs on
EXPOSE 8000

# Command to run the application
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "backend.wsgi"]
```

Django REST API, Docker and Snowflake Container

5. README.md

```
# Django REST API on Snowflake Container Services

This project deploys a Django REST API on Snowflake Container Services.

## Prerequisites

- Snowflake account with ACCOUNTADMIN privileges
- SnowSQL CLI installed
- Docker installed locally

## Setup Instructions

### 1. Initialize Snowflake Environment

```bash
snowsql -a <account_identifier> -u <username>
```

Then run the `setup.sql` script.

2. Build and Push Docker Image

```
# Build the image
docker build -t django-api .

# Tag the image
docker tag django-api <your_snowflake_account>.registry.snowflakecomputing.com/DJANGO_API/CONTAINERS/IMAGES/DJANGO_API_IMAGE

# Log in to Snowflake registry
docker login <your_snowflake_account>.registry.snowflakecomputing.com -u <username>

# Push the image
docker push <your_snowflake_account>.registry.snowflakecomputing.com/DJANGO_API/CONTAINERS/IMAGES/DJANGO_API_IMAGE
```

3. Deploy the Container Service

```
# Create the service
```
```

Django REST API, Docker and Snowflake Container

```
snowsql -a <account_identifier> -u <username> -q "CREATE SERVICE django-api I
N DATABASE DJANGO_API SCHEMA CONTAINERS FROM SPECIFICATION FILE @~/manifest.y
ml"

# Check service status
snowsql -a <account_identifier> -u <username> -q "SHOW SERVICES IN DATABASE D
JANGO_API SCHEMA CONTAINERS"
```

4. Access the API

Once deployed, you can access:

- API endpoints: `https://<service-url>/api/tasks/`
- Admin interface: `https://<service-url>/admin/`

Configuration

Update these environment variables in `manifest.yml`:

- `DJANGO_SECRET_KEY`: A secure secret key for Django
- `ALLOWED_HOSTS`: Comma-separated list of allowed hosts
- `CORS_ALLOWED_ORIGINS`: Comma-separated list of allowed origins

Maintenance

To update the service:

1. Push a new Docker image
2. Run:
   ```
   ALTER SERVICE django-api REFRESH;
   ```

To check logs:

```
SELECT SYSTEM$GET_SERVICE_LOGS('django-api', '0', 'django-api');

## Additional Notes
```

Django REST API, Docker and Snowflake Container

```
1. **Database Configuration**: You'll need to configure Django to use Snowfla
ke as the database backend. Install `django-snowflake` and update your settin
gs:

```python
DATABASES = {
    'default': {
        'ENGINE': 'django_snowflake',
        'NAME': os.getenv('DATABASE_NAME'),
        'SCHEMA': os.getenv('DATABASE_SCHEMA'),
        'ACCOUNT': os.getenv('SNOWFLAKE_ACCOUNT'),
        'USER': os.getenv('SNOWFLAKE_USER'),
        'PASSWORD': os.getenv('SNOWFLAKE_PASSWORD'),
        'WAREHOUSE': os.getenv('SNOWFLAKE_WAREHOUSE'),
    }
}
```

2. **Static Files**: For production, consider using Snowflake stages or an external service for static files.
3. **Scaling**: Adjust the `resources` section in `manifest.yml` based on your workload requirements.
```