**Apache Superset Complete Code Structure Documentation**

**Overview**

Apache Superset is a modern, enterprise-ready business intelligence web application that provides a comprehensive platform for data visualization and exploration. This documentation provides a complete understanding of Superset's architecture, code structure, and implementation details essential for customization, deployment, and development.

**Table of Contents**

1. [Architecture Overview](#architecture-overview)
2. [Repository Structure](#repository-structure)
3. [Backend Structure](#backend-structure)
4. [Frontend Structure](#frontend-structure)
5. [Database and Models](#database-and-models)
6. [Security and Authentication](#security-and-authentication)
7. [API Layer](#api-layer)
8. [Configuration System](#configuration-system)
9. [Plugin Architecture](#plugin-architecture)
10. [Deployment Guide](#deployment-guide)
11. [EC2 Deployment](#ec2-deployment)
12. [PostgreSQL Integration](#postgresql-integration)
13. [AWS SSO Integration](#aws-sso-integration)
14. [Customization Guide](#customization-guide)
15. [Development Environment](#development-environment)

**Architecture Overview**

**Core Components**

Apache Superset follows a modular architecture with several key components:

**Application Layer**

* **Flask Backend**: Python-based web server using Flask framework
* **React Frontend**: Modern TypeScript/React-based user interface
* **API Layer**: RESTful APIs for data access and management
* **Metadata Database**: Stores charts, dashboards, users, and configurations
* **Caching Layer**: Optional Redis/Memcached for performance optimization
* **Task Queue**: Celery workers for background processing
* **WebSocket Service**: Real-time communication support

**Data Integration**

* **SQL Alchemy**: Database connectivity and ORM
* **Database Connectors**: Support for 40+ database types
* **Query Engine**: SQL query processing and optimization
* **Semantic Layer**: Custom dimensions and metrics definitions

**Technology Stack**

**Backend Technologies:**

* Python 3.9+
* Flask web framework
* SQLAlchemy ORM
* Celery task queue
* Redis/PostgreSQL for storage
* Flask-AppBuilder for admin interface

**Frontend Technologies:**

* TypeScript (44.5% of codebase)
* React.js framework
* Redux for state management
* Ant Design component library
* Emotion for CSS-in-JS styling
* Webpack for bundling

**Repository Structure**

The Apache Superset repository is organized as a monorepo with the following structure:

apache/superset/  
├── .github/ # GitHub Actions workflows and templates  
├── .devcontainer/ # VS Code development container config  
├── ASF/ # Apache Software Foundation files  
├── CHANGELOG/ # Release changelog files  
├── RELEASING/ # Release process documentation  
├── RESOURCES/ # Additional resources and guides  
├── docker/ # Docker configuration and scripts  
├── docs/ # Documentation source files  
├── helm/superset/ # Kubernetes Helm charts  
├── requirements/ # Python dependency files  
├── scripts/ # Build and utility scripts  
├── superset-embedded-sdk/ # Embedding SDK for external apps  
├── superset-frontend/ # React/TypeScript frontend code  
├── superset-websocket/ # WebSocket service implementation  
├── superset/ # Python backend application code  
├── tests/ # Test suites (unit, integration, e2e)  
├── docker-compose\*.yml # Docker Compose configurations  
├── setup.py # Python package setup  
└── pyproject.toml # Python project configuration

**Key Directories Explained**

**docker/** - Contains all Docker-related configurations:

* docker-entrypoint-initdb.d/ - Database initialization scripts
* entrypoints/ - Container entry point scripts
* pythonpath\_dev/ - Development Python configuration
* docker-bootstrap.sh - Main bootstrap script
* .env - Environment variables

**superset-frontend/** - Complete frontend application:

* src/ - Source code directory
* packages/ - NPM packages and UI components
* cypress-base/ - End-to-end testing
* webpack.config.js - Build configuration
* package.json - Node.js dependencies

**superset/** - Backend Python application:

* models/ - SQLAlchemy database models
* views/ - Flask view controllers
* security/ - Authentication and authorization
* connectors/ - Database connector implementations
* charts/ - Chart and visualization logic
* dashboards/ - Dashboard management
* sql\_lab/ - SQL Editor functionality

**Backend Structure**

**Core Python Modules**

The backend is structured following Flask application patterns:

superset/  
├── \_\_init\_\_.py # Application factory  
├── app.py # Flask app creation and configuration  
├── config.py # Default configuration settings  
├── extensions.py # Flask extensions initialization  
├── models/ # Database models  
│ ├── core.py # Core models (Database, Table, etc.)  
│ ├── dashboard.py # Dashboard models  
│ ├── slice.py # Chart/slice models  
│ ├── sql\_lab.py # SQL Lab models  
│ └── annotations.py # Annotation models  
├── views/ # Flask views and controllers  
│ ├── base.py # Base view classes  
│ ├── core.py # Core application views  
│ ├── dashboard/ # Dashboard views  
│ ├── chart/ # Chart views  
│ └── sql\_lab/ # SQL Lab views  
├── security/ # Security and authentication  
│ ├── manager.py # Security manager  
│ └── views.py # Security-related views  
├── connectors/ # Database connectors  
│ ├── sqla/ # SQL Alchemy connector  
│ └── base.py # Base connector interface  
├── db\_engine\_specs/ # Database-specific implementations  
├── commands/ # Command pattern implementations  
├── daos/ # Data Access Objects  
├── tasks/ # Celery background tasks  
└── utils/ # Utility functions

**Key Backend Components**

**Application Factory (app.py)**  
Creates and configures the Flask application instance with extensions and blueprints.

**Models (models/)**

* SQLAlchemy models representing database entities
* Relationships between users, dashboards, charts, and databases
* Audit logging and metadata tracking

**Security Manager (security/manager.py)**

* Handles authentication and authorization
* Role-based access control (RBAC)
* Integration points for custom authentication

**Database Connectors (connectors/)**

* Abstraction layer for different database types
* SQL query generation and execution
* Schema introspection and metadata extraction

**DB Engine Specs (db\_engine\_specs/)**

* Database-specific optimizations and features
* Custom SQL dialect handling
* Performance tuning for different engines

**Frontend Structure**

**React Application Architecture**

The frontend is a sophisticated TypeScript/React application:

superset-frontend/  
├── src/ # Main source directory  
│ ├── components/ # Reusable React components  
│ │ ├── Chart/ # Chart components  
│ │ ├── Dashboard/ # Dashboard components   
│ │ ├── MessageToasts/ # Notification system  
│ │ └── Modal/ # Modal dialogs  
│ ├── explore/ # Chart building interface  
│ ├── dashboard/ # Dashboard functionality  
│ ├── SqlLab/ # SQL Editor interface  
│ ├── views/ # Main application views  
│ ├── utils/ # Utility functions  
│ ├── hooks/ # Custom React hooks  
│ ├── types/ # TypeScript type definitions  
│ └── setupApp.tsx # Application bootstrapping  
├── packages/ # Monorepo packages  
│ ├── superset-ui-chart-controls/ # Chart controls  
│ ├── superset-ui-core/ # Core UI utilities  
│ └── generator-superset/ # Plugin generator  
├── cypress-base/ # E2E testing framework  
├── spec/ # Frontend unit tests  
├── stylesheets/ # Legacy CSS styles  
├── webpack.config.js # Build configuration  
└── package.json # Dependencies and scripts

**Key Frontend Concepts**

**Component Organization**

* Components are organized by feature and reusability
* Each component has its own directory with test and story files
* TypeScript interfaces define component props and state

**State Management**

* Redux for global application state
* React hooks for local component state
* Async actions for API communication

**Styling System**

* Emotion CSS-in-JS for component styling
* Ant Design component library as base
* Theme system for consistent design

**Plugin System**

* Modular visualization plugins
* Dynamic plugin loading and registration
* Custom plugin development support

**Database and Models**

**Core Models**

**Database Model**

class Database(Model, AuditMixinNullable, ImportExportMixin):  
 """Represents a database connection"""  
 database\_name = Column(String(250), unique=True, nullable=False)  
 sqlalchemy\_uri = Column(String(1024), nullable=False)  
 created\_by\_fk = Column(Integer, ForeignKey('ab\_user.id'))  
 changed\_by\_fk = Column(Integer, ForeignKey('ab\_user.id'))

**Dashboard Model**

class Dashboard(Model, AuditMixinNullable, ImportExportMixin):  
 """Dashboard model"""  
 dashboard\_title = Column(String(500))  
 position\_json = Column(MediumText())  
 css = Column(MediumText())  
 json\_metadata = Column(Text())  
 published = Column(Boolean, default=False)

**Chart/Slice Model**

class Slice(Model, AuditMixinNullable, ImportExportMixin):  
 """Chart/visualization model"""  
 slice\_name = Column(String(250), nullable=False)  
 datasource\_id = Column(Integer)  
 datasource\_type = Column(String(200))  
 params = Column(Text)  
 query\_context = Column(Text)

**Metadata Database Schema**

The metadata database stores:

* User accounts and roles
* Database connections
* Charts and dashboards
* Query logs and audit trails
* Application configurations
* Cached query results

**Database Migrations**

Superset uses Alembic for database migrations:

* Migration files in superset/migrations/versions/
* Automatic schema updates during upgrades
* Support for multiple database backends

**Security and Authentication**

**Authentication Mechanisms**

**Built-in Authentication**

* Database-stored user accounts
* Password-based authentication
* Session management

**LDAP Integration**

AUTH\_TYPE = AUTH\_LDAP  
AUTH\_LDAP\_SERVER = "ldap://ldapserver.new"  
AUTH\_LDAP\_USE\_TLS = False  
AUTH\_LDAP\_BIND\_USER = "cn=admin,dc=example,dc=com"  
AUTH\_LDAP\_BIND\_PASSWORD = "admin\_password"

**OAuth Integration**

from flask\_appbuilder.security.manager import AUTH\_OAUTH  
AUTH\_TYPE = AUTH\_OAUTH  
OAUTH\_PROVIDERS = [{  
 'name': 'google',  
 'token\_key': 'access\_token',  
 'icon': 'fa-google',  
 'remote\_app': {  
 'client\_id': 'GOOGLE\_CLIENT\_ID',  
 'client\_secret': 'GOOGLE\_CLIENT\_SECRET',  
 'server\_metadata\_url': 'https://accounts.google.com/.well-known/openid\_configuration',  
 'client\_kwargs': {'scope': 'openid email profile'}  
 }  
}]

**Role-Based Access Control (RBAC)**

**Default Roles**

* Admin: Full system access
* Alpha: Can edit all dashboards
* Gamma: Can view assigned dashboards
* sql\_lab: Can use SQL Lab

**Row Level Security (RLS)**

RLS\_BASE\_TRANSFORMATIONS = {  
 "{{current\_user\_id()}}": lambda: g.user.id,  
 "{{current\_username()}}": lambda: g.user.username,  
}

**API Layer**

**REST API Structure**

Superset provides comprehensive REST APIs:

**Authentication APIs**

* /api/v1/security/login - User authentication
* /api/v1/security/refresh - Token refresh
* /api/v1/security/csrf\_token - CSRF token

**Chart APIs**

* /api/v1/chart/ - Chart CRUD operations
* /api/v1/chart/{id}/data - Chart data
* /api/v1/chart/{id}/cache\_screenshot - Chart screenshots

**Dashboard APIs**

* /api/v1/dashboard/ - Dashboard management
* /api/v1/dashboard/{id}/charts - Dashboard charts
* /api/v1/dashboard/{id}/filter\_sets - Filter configurations

**Database APIs**

* /api/v1/database/ - Database connections
* /api/v1/database/{id}/schemas - Schema information
* /api/v1/database/{id}/table/{table}/select\_star - Table preview

**Custom API Development**

**Creating Custom Endpoints**

from flask\_appbuilder import ModelRestApi  
from flask\_appbuilder.security.decorators import permission\_name  
  
class CustomModelRestApi(ModelRestApi):  
 resource\_name = 'custom'  
 allow\_browser\_login = True  
   
 @permission\_name('read')  
 @expose('/custom\_endpoint')  
 def custom\_endpoint(self):  
 return self.response(200, message="Custom endpoint")  
  
# Register the API  
appbuilder.add\_api(CustomModelRestApi)

**Configuration System**

**Configuration Files**

**Main Configuration (superset/config.py)**  
Contains default settings that can be overridden.

**Custom Configuration (superset\_config.py)**  
User-defined settings placed in Python path.

**Environment Variables**  
Many settings can be controlled via environment variables:

SUPERSET\_CONFIG\_PATH=/path/to/superset\_config.py  
DATABASE\_URL=postgresql://user:pass@localhost/superset  
REDIS\_URL=redis://localhost:6379/0

**Key Configuration Areas**

**Database Configuration**

SQLALCHEMY\_DATABASE\_URI = 'postgresql://user:pass@host:port/db'  
SQLALCHEMY\_ENGINE\_OPTIONS = {  
 'pool\_size': 10,  
 'pool\_recycle': 3600,  
 'pool\_pre\_ping': True,  
}

**Caching Configuration**

CACHE\_CONFIG = {  
 'CACHE\_TYPE': 'redis',  
 'CACHE\_DEFAULT\_TIMEOUT': 300,  
 'CACHE\_KEY\_PREFIX': 'superset\_',  
 'CACHE\_REDIS\_URL': 'redis://localhost:6379/0'  
}

**Security Configuration**

SECRET\_KEY = 'your-secret-key-here'  
WTF\_CSRF\_ENABLED = True  
WTF\_CSRF\_TIME\_LIMIT = None  
PERMANENT\_SESSION\_LIFETIME = timedelta(days=31)

**Plugin Architecture**

**Visualization Plugins**

**Plugin Structure**

superset-plugin-chart-custom/  
├── src/  
│ ├── plugin/  
│ │ ├── buildQuery.ts  
│ │ ├── controlPanel.ts  
│ │ ├── index.ts  
│ │ └── transformProps.ts  
│ └── components/  
│ └── CustomChart.tsx  
├── package.json  
└── README.md

**Plugin Registration**

// In superset-frontend/src/visualizations/presets/MainPreset.js  
import CustomChartPlugin from 'superset-plugin-chart-custom';  
  
new CustomChartPlugin().configure({ key: 'custom-chart' })

**Control Panel Definition**

export default {  
 controlPanelSections: [  
 {  
 label: t('Query'),  
 expanded: true,  
 controlSetRows: [  
 ['metrics'],  
 ['groupby'],  
 ['limit'],  
 ],  
 },  
 ],  
};

**Database Connector Plugins**

**Custom Database Connector**

from superset.db\_engine\_specs.base import BaseEngineSpec  
  
class CustomDBEngineSpec(BaseEngineSpec):  
 engine = 'custom\_db'  
 engine\_name = 'Custom Database'  
   
 @classmethod  
 def get\_schema\_names(cls, database):  
 # Custom schema introspection logic  
 return ['schema1', 'schema2']

**Deployment Guide**

**Docker Deployment**

**Basic Docker Compose Setup**

version: '3.8'  
services:  
 redis:  
 image: redis:latest  
 restart: unless-stopped  
   
 db:  
 env\_file: docker/.env-non-dev  
 image: postgres:13  
 restart: unless-stopped  
 volumes:  
 - db\_home:/var/lib/postgresql/data  
   
 superset:  
 env\_file: docker/.env-non-dev  
 image: apache/superset:latest  
 restart: unless-stopped  
 depends\_on:  
 - db  
 - redis  
 ports:  
 - 8088:8088  
 volumes:  
 - ./docker/pythonpath\_dev:/app/pythonpath

**Production Docker Configuration**

services:  
 superset:  
 image: apache/superset:latest  
 environment:  
 - SUPERSET\_ENV=production  
 - DATABASE\_URL=postgresql://user:pass@db:5432/superset  
 - REDIS\_URL=redis://redis:6379/0  
 deploy:  
 replicas: 3  
 resources:  
 limits:  
 memory: 2G  
 cpus: '1'

**Kubernetes Deployment**

**Using Helm Charts**

helm repo add superset https://apache.github.io/superset  
helm upgrade --install superset superset/superset \  
 --set postgresql.enabled=true \  
 --set redis.enabled=true \  
 --set supersetNode.enabled=false

**Custom Values**

# values.yaml  
postgresql:  
 enabled: true  
 auth:  
 postgresPassword: superset  
 database: superset  
  
redis:  
 enabled: true  
  
configOverrides:  
 secret: |  
 SECRET\_KEY = 'your-secret-key'  
 SQLALCHEMY\_DATABASE\_URI = 'postgresql://...'

**EC2 Deployment**

**Prerequisites**

**System Requirements**

# Update system  
sudo yum update -y # Amazon Linux  
sudo apt update # Ubuntu  
  
# Install Python 3.8+  
sudo yum install python3 python3-pip python3-dev -y  
sudo apt install python3 python3-pip python3-dev -y  
  
# Install Node.js 16+  
curl -fsSL https://deb.nodesource.com/setup\_16.x | sudo -E bash -  
sudo apt-get install -y nodejs  
  
# Install system dependencies  
sudo apt install build-essential libssl-dev libffi-dev \  
 python3-dev libsasl2-dev libldap2-dev \  
 default-libmysqlclient-dev

**Installation Steps**

**1. Create Virtual Environment**

python3 -m venv superset\_env  
source superset\_env/bin/activate  
pip install --upgrade pip setuptools

**2. Install Superset**

pip install apache-superset psycopg2-binary redis celery

**3. Initialize Database**

# Set FLASK\_APP  
export FLASK\_APP=superset  
  
# Create admin user  
superset fab create-admin \  
 --username admin \  
 --firstname Superset \  
 --lastname Admin \  
 --email admin@superset.com \  
 --password admin  
  
# Initialize database  
superset db upgrade  
  
# Setup roles and permissions  
superset init  
  
# Load example data (optional)  
superset load\_examples

**4. Configuration File**

# /home/ubuntu/superset\_config.py  
import os  
  
# Database configuration  
SQLALCHEMY\_DATABASE\_URI = 'postgresql://superset:password@localhost:5432/superset'  
  
# Redis for caching and celery  
CACHE\_CONFIG = {  
 'CACHE\_TYPE': 'redis',  
 'CACHE\_DEFAULT\_TIMEOUT': 300,  
 'CACHE\_KEY\_PREFIX': 'superset\_',  
 'CACHE\_REDIS\_URL': 'redis://localhost:6379/0'  
}  
  
# Celery configuration  
class CeleryConfig(object):  
 BROKER\_URL = 'redis://localhost:6379/0'  
 CELERY\_IMPORTS = ('superset.sql\_lab', 'superset.tasks')  
 CELERY\_RESULT\_BACKEND = 'redis://localhost:6379/0'  
 CELERY\_ANNOTATIONS = {'tasks.add': {'rate\_limit': '10/s'}}  
  
CELERY\_CONFIG = CeleryConfig  
  
# Security  
SECRET\_KEY = 'your-secret-key-change-this'  
WTF\_CSRF\_ENABLED = True  
  
# Feature flags  
FEATURE\_FLAGS = {  
 'ALERTS\_REPORTS': True,  
 'DASHBOARD\_FILTERS\_EXPERIMENTAL': True,  
 'ENABLE\_TEMPLATE\_PROCESSING': True,  
}  
  
# Email configuration (optional)  
SMTP\_HOST = 'localhost'  
SMTP\_STARTTLS = True  
SMTP\_SSL = False  
SMTP\_USER = 'superset'  
SMTP\_PORT = 25  
SMTP\_PASSWORD = 'superset'  
SMTP\_MAIL\_FROM = 'superset@example.com'

**5. Systemd Service Files**

**Superset Web Service (/etc/systemd/system/superset.service)**

[Unit]  
Description=Apache Superset Web Server  
After=network.target  
  
[Service]  
Type=notify  
User=ubuntu  
Group=ubuntu  
Environment=PYTHONPATH=/home/ubuntu  
Environment=SUPERSET\_CONFIG\_PATH=/home/ubuntu/superset\_config.py  
ExecStart=/home/ubuntu/superset\_env/bin/gunicorn \  
 --bind 0.0.0.0:8088 \  
 --workers 4 \  
 --worker-class gevent \  
 --timeout 60 \  
 --keep-alive 2 \  
 --max-requests 1000 \  
 --max-requests-jitter 100 \  
 --preload \  
 "superset.app:create\_app()"  
Restart=always  
RestartSec=10  
  
[Install]  
WantedBy=multi-user.target

**Celery Worker Service (/etc/systemd/system/superset-worker.service)**

[Unit]  
Description=Apache Superset Celery Worker  
After=network.target  
  
[Service]  
Type=forking  
User=ubuntu  
Group=ubuntu  
Environment=PYTHONPATH=/home/ubuntu  
Environment=SUPERSET\_CONFIG\_PATH=/home/ubuntu/superset\_config.py  
ExecStart=/home/ubuntu/superset\_env/bin/celery \  
 --app=superset.tasks.celery\_app:app worker \  
 --loglevel=info  
Restart=always  
RestartSec=10  
  
[Install]  
WantedBy=multi-user.target

**6. Nginx Configuration**

# /etc/nginx/sites-available/superset  
server {  
 listen 80;  
 server\_name your-domain.com;  
  
 location / {  
 proxy\_pass http://127.0.0.1:8088;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 proxy\_redirect off;  
 }  
}

**7. Start Services**

sudo systemctl daemon-reload  
sudo systemctl enable superset superset-worker  
sudo systemctl start superset superset-worker  
sudo systemctl enable nginx  
sudo systemctl start nginx

**PostgreSQL Integration**

**Database Setup**

**1. Install PostgreSQL**

sudo apt install postgresql postgresql-contrib  
sudo systemctl start postgresql  
sudo systemctl enable postgresql

**2. Create Database and User**

sudo -u postgres psql  
  
CREATE DATABASE superset;  
CREATE USER superset WITH ENCRYPTED PASSWORD 'your-password';  
GRANT ALL PRIVILEGES ON DATABASE superset TO superset;  
ALTER USER superset CREATEDB;  
\q

**3. Configure Connection**

# In superset\_config.py  
SQLALCHEMY\_DATABASE\_URI = 'postgresql://superset:your-password@localhost:5432/superset'  
  
# Connection pool settings  
SQLALCHEMY\_ENGINE\_OPTIONS = {  
 'pool\_size': 10,  
 'pool\_recycle': 3600,  
 'pool\_pre\_ping': True,  
 'max\_overflow': 20,  
}

**Adding Custom Data Sources**

**1. Install Database Drivers**

# PostgreSQL  
pip install psycopg2-binary  
  
# MySQL  
pip install mysqlclient  
  
# SQL Server  
pip install pymssql  
  
# Oracle  
pip install cx\_Oracle  
  
# Snowflake  
pip install snowflake-sqlalchemy

**2. Configure Data Source**

# In Superset UI or via configuration  
DATABASE\_CONFIG = {  
 'name': 'Production PostgreSQL',  
 'sqlalchemy\_uri': 'postgresql://user:pass@host:port/database',  
 'expose\_in\_sqllab': True,  
 'allow\_ctas': True,  
 'allow\_cvas': True,  
 'allow\_dml': False,  
}

**3. Test Connection**

from superset.models.core import Database  
from superset import db  
  
# Test database connection  
database = Database(  
 database\_name='test\_connection',  
 sqlalchemy\_uri='postgresql://user:pass@host:port/db'  
)  
  
# Test connection  
try:  
 database.get\_sqla\_engine().connect()  
 print("Connection successful!")  
except Exception as e:  
 print(f"Connection failed: {e}")

**AWS SSO Integration**

**SAML Configuration**

**1. AWS SSO Setup**

# In superset\_config.py  
from flask\_appbuilder.security.manager import AUTH\_OAUTH  
  
AUTH\_TYPE = AUTH\_OAUTH  
AUTH\_USER\_REGISTRATION = True  
AUTH\_USER\_REGISTRATION\_ROLE = "Gamma"  
  
# SAML Configuration  
from flask\_appbuilder.security.manager import AUTH\_OID  
AUTH\_TYPE = AUTH\_OID  
OIDC\_CLIENT\_SECRETS = '/path/to/client\_secrets.json'  
OIDC\_ID\_TOKEN\_COOKIE\_SECURE = False  
OIDC\_REQUIRE\_VERIFIED\_EMAIL = False

**2. AWS Cognito Integration**

# Cognito OAuth configuration  
OAUTH\_PROVIDERS = [{  
 'name': 'aws\_cognito',  
 'token\_key': 'access\_token',  
 'icon': 'fa-aws',  
 'remote\_app': {  
 'client\_id': 'YOUR\_COGNITO\_CLIENT\_ID',  
 'client\_secret': 'YOUR\_COGNITO\_CLIENT\_SECRET',  
 'api\_base\_url': 'https://YOUR\_DOMAIN.auth.us-east-1.amazoncognito.com/',  
 'access\_token\_url': 'https://YOUR\_DOMAIN.auth.us-east-1.amazoncognito.com/oauth2/token',  
 'authorize\_url': 'https://YOUR\_DOMAIN.auth.us-east-1.amazoncognito.com/oauth2/authorize',  
 'request\_token\_url': None,  
 'client\_kwargs': {  
 'scope': 'email openid profile'  
 }  
 }  
}]

**3. Custom Security Manager**

from superset.security import SupersetSecurityManager  
  
class CustomSecurityManager(SupersetSecurityManager):  
 def oauth\_user\_info(self, provider, response=None):  
 if provider == 'aws\_cognito':  
 # Extract user info from Cognito response  
 me = self.appbuilder.sm.oauth\_remotes[provider].get('userinfo')  
 return {  
 'username': me.data.get('preferred\_username', ''),  
 'first\_name': me.data.get('given\_name', ''),  
 'last\_name': me.data.get('family\_name', ''),  
 'email': me.data.get('email', '')  
 }  
 return {}  
  
CUSTOM\_SECURITY\_MANAGER = CustomSecurityManager

**Row Level Security with AWS**

**1. Configure RLS**

# Custom RLS transformations  
RLS\_BASE\_TRANSFORMATIONS = {  
 "{{current\_user\_id()}}": lambda: g.user.id,  
 "{{current\_username()}}": lambda: g.user.username,  
 "{{aws\_user\_groups()}}": lambda: get\_aws\_user\_groups(g.user),  
}  
  
def get\_aws\_user\_groups(user):  
 # Fetch user groups from AWS SSO  
 # Implementation depends on your AWS setup  
 return ['group1', 'group2']

**2. Apply RLS Filters**

-- In dataset SQL filter  
SELECT \* FROM sales\_data   
WHERE region IN ({{aws\_user\_groups()}})

**Customization Guide**

**Frontend Customization**

**1. Custom Branding**

// In superset-frontend/src/assets/branding.ts  
export const supersetTheme = {  
 colors: {  
 primary: '#1890ff',  
 secondary: '#f50',  
 success: '#52c41a',  
 warning: '#faad14',  
 error: '#f5222d',  
 },  
 typography: {  
 families: {  
 sansSerif: '"Inter", "Helvetica Neue", Helvetica, Arial, sans-serif',  
 monospace: '"Fira Code", "Courier New", monospace',  
 },  
 },  
};

**2. Custom Components**

// Custom header component  
const CustomHeader: React.FC = () => {  
 return (  
 <div className="custom-header">  
 <img src="/static/assets/images/custom-logo.png" alt="Company Logo" />  
 <h1>Company BI Platform</h1>  
 </div>  
 );  
};  
  
// Register custom component  
export default CustomHeader;

**3. CSS Customizations**

/\* In superset-frontend/src/assets/stylesheets/custom.css \*/  
.navbar-brand {  
 background-image: url('/static/assets/images/your-logo.png');  
 background-size: contain;  
 background-repeat: no-repeat;  
}  
  
.dashboard-header {  
 background: linear-gradient(90deg, #667eea 0%, #764ba2 100%);  
 color: white;  
}

**Backend Customization**

**1. Custom Views**

from flask\_appbuilder import ModelView  
from flask\_appbuilder.models.sqla.interface import SQLAInterface  
  
class CustomModelView(ModelView):  
 datamodel = SQLAInterface(YourModel)  
 list\_columns = ['name', 'created\_on', 'created\_by']  
 search\_columns = ['name']  
   
 @expose('/custom\_action')  
 def custom\_action(self):  
 # Custom business logic  
 return self.render\_template('custom\_template.html')  
  
# Register the view  
appbuilder.add\_view(CustomModelView, "Custom View", category="Custom")

**2. Custom SQL Functions**

from superset.db\_engine\_specs.base import BaseEngineSpec  
  
class CustomEngineSpec(BaseEngineSpec):  
 engine = 'postgresql'  
   
 @classmethod  
 def get\_function\_names(cls, database):  
 # Add custom SQL functions  
 functions = super().get\_function\_names(database)  
 functions.extend(['custom\_function', 'another\_function'])  
 return functions

**3. Custom Metrics**

# Custom metric definitions  
CUSTOM\_METRICS = [  
 {  
 'metric\_name': 'revenue\_per\_user',  
 'verbose\_name': 'Revenue per User',  
 'metric\_type': 'simple',  
 'expression': 'SUM(revenue) / COUNT(DISTINCT user\_id)'  
 }  
]

**Plugin Development**

**1. Create Visualization Plugin**

# Using the plugin generator  
npm install -g yo  
npm install -g @superset-ui/generator-superset  
  
# Generate new plugin  
yo @superset-ui/superset

**2. Plugin Implementation**

import { ChartDataResponseResult, ChartProps } from '@superset-ui/core';  
import React from 'react';  
  
interface CustomChartProps extends ChartProps {  
 data: ChartDataResponseResult[];  
}  
  
const CustomChart: React.FC<CustomChartProps> = ({ data, width, height }) => {  
 return (  
 <div style={{ width, height }}>  
 {/\* Custom visualization logic \*/}  
 <h3>Custom Chart</h3>  
 <pre>{JSON.stringify(data, null, 2)}</pre>  
 </div>  
 );  
};  
  
export default CustomChart;

**Development Environment**

**Local Development Setup**

**1. Prerequisites**

# Install Node.js 16+  
nvm install 16  
nvm use 16  
  
# Install Python 3.9+  
pyenv install 3.9.0  
pyenv local 3.9.0  
  
# Install system dependencies  
sudo apt install build-essential libssl-dev libffi-dev \  
 python3-dev libsasl2-dev libldap2-dev

**2. Clone and Setup**

git clone https://github.com/apache/superset.git  
cd superset  
  
# Setup Python environment  
python3 -m venv venv  
source venv/bin/activate  
pip install -r requirements/development.txt  
pip install -e .  
  
# Setup pre-commit hooks  
pre-commit install  
  
# Initialize database  
export FLASK\_APP=superset  
superset db upgrade  
superset fab create-admin  
superset init  
superset load\_examples

**3. Frontend Development**

cd superset-frontend  
npm ci  
  
# Development server with hot reload  
npm run dev-server  
  
# Build for production  
npm run build

**4. Running Tests**

# Python tests  
pytest tests/  
  
# Frontend tests  
cd superset-frontend  
npm test  
  
# E2E tests  
npm run cypress:run

**Docker Development**

**1. Development with Docker Compose**

# Start all services  
docker-compose up  
  
# Access Superset at http://localhost:8088  
# Default credentials: admin/admin

**2. Custom Development Configuration**

# docker-compose.override.yml  
version: '3.8'  
services:  
 superset:  
 volumes:  
 - ./custom\_config.py:/app/pythonpath/superset\_config.py  
 - ./custom\_assets:/app/superset/static/assets/custom

**Debugging**

**1. Backend Debugging**

# VSCode launch.json  
{  
 "version": "0.2.0",  
 "configurations": [  
 {  
 "name": "Python: Flask",  
 "type": "python",  
 "request": "launch",  
 "program": "${workspaceFolder}/superset/app.py",  
 "env": {  
 "FLASK\_ENV": "development",  
 "SUPERSET\_CONFIG\_PATH": "${workspaceFolder}/superset\_config.py"  
 },  
 "args": ["run", "-p", "8088", "--with-threads", "--reload", "--debugger"],  
 "console": "internalConsole"  
 }  
 ]  
}

**2. Frontend Debugging**

# Enable source maps  
npm run dev-server  
  
# Debug in browser dev tools  
# React DevTools extension recommended

**3. Database Debugging**

# Enable SQL logging  
SQLALCHEMY\_ENGINE\_OPTIONS = {  
 'echo': True # Logs all SQL queries  
}  
  
# Query profiling  
ENABLE\_SUPERSET\_META\_DB = True

**Troubleshooting**

**Common Issues**

**1. Database Connection Issues**

# Check database connectivity  
ping your-database-host  
  
# Test connection string  
python -c "from sqlalchemy import create\_engine; engine = create\_engine('your-connection-string'); engine.connect()"  
  
# Check firewall rules  
sudo ufw status

**2. Frontend Build Issues**

# Clear npm cache  
npm cache clean --force  
  
# Remove node\_modules and reinstall  
rm -rf node\_modules package-lock.json  
npm install  
  
# Check Node.js version  
node --version # Should be 16+

**3. Permission Issues**

# Check file permissions  
ls -la /path/to/superset/  
  
# Fix ownership  
sudo chown -R ubuntu:ubuntu /path/to/superset/  
  
# Check Python path  
python -c "import sys; print(sys.path)"

**4. Memory Issues**

# Check memory usage  
free -h  
  
# Monitor processes  
top  
htop  
  
# Adjust worker processes  
# In gunicorn config: --workers 2

**Performance Optimization**

**1. Database Optimization**

-- Create indexes for common queries  
CREATE INDEX idx\_dashboards\_published ON dashboards(published);  
CREATE INDEX idx\_slices\_datasource ON slices(datasource\_id, datasource\_type);

**2. Caching Configuration**

# Redis caching  
CACHE\_CONFIG = {  
 'CACHE\_TYPE': 'redis',  
 'CACHE\_DEFAULT\_TIMEOUT': 86400, # 24 hours  
 'CACHE\_KEY\_PREFIX': 'superset\_',  
 'CACHE\_REDIS\_URL': 'redis://localhost:6379/0'  
}  
  
# Enable chart caching  
ENABLE\_SUPERSET\_CHART\_CACHE = True

**3. Query Optimization**

# Limit query results  
QUERY\_SEARCH\_LIMIT = 1000  
SAMPLES\_ROW\_LIMIT = 50000  
  
# Query timeout  
SQLLAB\_TIMEOUT = 300 # 5 minutes

**Conclusion**

This comprehensive documentation provides a complete understanding of Apache Superset's architecture, code structure, and implementation details. It covers everything needed for:

* Understanding the codebase structure
* Setting up development environments
* Deploying to EC2 with PostgreSQL
* Integrating AWS SSO authentication
* Customizing frontend and backend components
* Developing custom plugins and extensions
* Troubleshooting common issues

With this knowledge, you should be able to successfully deploy, customize, and extend Apache Superset to meet your specific business intelligence requirements.

For additional support and community resources:

* Official Documentation: <https://superset.apache.org/docs/>
* GitHub Repository: <https://github.com/apache/superset>
* Community Slack: <https://join.slack.com/t/apache-superset/shared_invite/>
* Stack Overflow: Tag questions with apache-superset

Remember to always follow security best practices when deploying to production environments, and keep your Superset installation updated with the latest security patches and features.