# Overview

Casdoor is a UI-first Identity Access Management (IAM) / Single-Sign-On (SSO) platform based on OAuth 2.0, OIDC, SAML and CAS.

You need to enable JavaScript to run this app.

Casdoor serves both the web UI and the login requests from the application users.

# Casdoor features:

1. Front-end and back-end separate architecture, developed by Golang, Casdoor supports high concurrency, provides web-based managing UI and supports multiple languages(Chinese, English).

2. Casdoor supports third-party applications login, such as GitHub, Google, QQ, WeChat, etc., and supports the extension of third-party login with plugins.

3. With Casbin based authorization management, Casdoor supports ACL, RBAC, ABAC, RESTful accessing control models.

4. Phone verification code, email verification code and password retrieval functions.

5. Accessing logs auditing and recording.

6. Alibaba Cloud, Tencent Cloud, Qiniu Cloud image CDN cloud storage.

7. Customizable registration, login, and password retrieval pages.

8. Casdoor supports integration with existing systems by db sync, so users can transition to Casdoor smoothly.

9. Casdoor supports mainstream databases: MySQL, PostgreSQL, SQL Server, etc., and supports the extension of new databases with plugins.

# How it works:

## Step 0 (Pre-knowledge)

1. Casdoor's authorization process is built upon the OAuth 2.0 protocol, therefore it's highly recommended to take a brief look at how indeed does OAuth 2.0 works. An introduction to OAuth 2.0.

## Step 1 (Authorization Request)

Your Application (could be a website or whatever) should compose an URL in this format `endpoint/login/oauth/authorize?client_id=xxx&response_type=code&redirect_uri=xxx&scope=read&state=xxx`. In the URL replace `endpoint` with your Casdoor's host URL, and replace `xxx` with your own info.

> HINTS
>
> How to fill out the `xxx` parts?
>
> - For `client_id`: you can find this under each single Application
> - For `redirect_uri`: you should set this to your own Application's callback URL, with this info, Casdoor can know where to send info back after authorization
> - For `state`: you should fill this out with your Application name

The Application will speak to the user: *"Hey, now I need some resources and I need your permission to take these resources, Do you mind going to this URL and filling out your*

*username and password for me?"*

With the correctly composed URL, your Application will make a user launch a request to this URL, and the `Authorization Request` is done.

## Step 2 (Authorization Grant)

This step is straightforward: the user is redirected to the URL composed above, and the user will see the login page from Casdoor. By typing the correct username and credential into the login page, Casdoor now knows the identity of the user, and is about to send two keywords: `code` and `state` back to the callback URL set in Step 1.

The user opens the URL and provides the credentials to Casdoor. Casdoor will say: *"Looking good ~ this is the user (who is authorizing the Application to get the `code` and `state`) I know in my database, and I will send the `code` and `state` back to the Application using the callback URL (`redirect_uri`)"*

With these two keywords sent back to your Application, the authorization is now granted to the app, and thus `Authorization Grant` is completed.

> TIP
>
> Casdoor also provides third-party logins, in this case, you will not see the credentials entry page but a list of third-party providers. You can login to your app using these providers, with Casdoor as a middle layer (middleware).

## Step 3 (Authorization Grant)

In this step, your Application already has the code in hand from Step 2, and it will speak to Casdoor: *"Hey, now the user agreed to give me the `code`, do you wanna check this `code` out and give me the `access_token`?"*

## Step 4 (Access Token)

In this step, Casdoor speaks back to the Application: *"You know what, this `code` seems legit, you must be the right Application. Come here, it's the `access_token`."* With this `code`,

Casdoor knows that it is an authorized Application (authorization given by the correct user in Step 2) trying to get the `access_token` (, which will be used later to get more useful things).

## Step 5 (Access Token)

In this step, your Application says: *"Nice one, just got the fresh-and-tasty `access_token`, I can now use it to get something more valuable from the `Resource Server`!"*

And your Application turns to the `Resource Server`: *"Hey buddy, wanna check this `access_token` out? I got this from Casdoor and do you want to see if this is the correct one you had with Casdoor?"*

## Step 6 (Protected Resource)

The `Resource Server` speaks back to your Application: *"Not bad ~ it seems just like the one I had with Casdoor, and Casdoor says whoever holds this `access_token` can have these `Protected Resources`. Now, you take it!"*

And this is basically how Casdoor works with your Application.

> HINT
>
> Casdoor can play both `Authorization Server` and `Resource Server` parts, i.e. Casdoor authorizes our Application to get resources (e.g. usually the current logged in user's info) from Casdoor's database.

# Online demo

## Casdoor

Here is an online demo deployed by Casbin.

- Casdoor official demo

Global admin login:

- Username: `admin`
- Password: `123`

## Casbin-OA

Casbin-OA is one of Casbin web apps. It uses Casdoor as authentication.

- Casbin-OA
- Source code: https://github.com/casbin/casbin-oa

## Casnode

Casnode is the official forum developed by Casbin community.

It uses Casdoor as authentication platform and manage members.

- Casnode
- Source code: https://github.com/casbin/casnode

# Architecture

Casdoor contains 2 parts:

| Name | Description | Language | Source code |
|---|---|---|---|
| Frontend | Web frontend UI for Casdoor | JavaScript + React | https://github.com/casdoor/casdoor/tree/master/web |
| Backend | RESTful API backend for Casdoor | Golang + Beego + SQL | https://github.com/casdoor/casdoor |

# Core Concepts

As Casdoor's administrator, you should get familiar with at least 4 core concepts: `Organization`, `User`, `Application` and `Provider`.

> **TIP**
> In the following parts, we will use the demo site: https://door.casdoor.com as example.

## Organization

In Casdoor, an organization is a container for users and applications. E.g., all the employees of a company or all the customers of a business can be abstracted as one organization. The Organization class definition is shown as follows:

```go
type Organization struct {
    Owner       string `xorm:"varchar(100) notnull pk" json:"owner"`
    Name        string `xorm:"varchar(100) notnull pk" json:"name"`
    CreatedTime string `xorm:"varchar(100)" json:"createdTime"`

    DisplayName        string   `xorm:"varchar(100)" json:"displayName"`
    WebsiteUrl         string   `xorm:"varchar(100)" json:"websiteUrl"`
    Favicon            string   `xorm:"varchar(100)" json:"favicon"`
    PasswordType       string   `xorm:"varchar(100)" json:"passwordType"`
    PasswordSalt       string   `xorm:"varchar(100)" json:"passwordSalt"`
    PhonePrefix        string   `xorm:"varchar(10)"  json:"phonePrefix"`
    DefaultAvatar      string   `xorm:"varchar(100)" json:"defaultAvatar"`
    Tags               []string `xorm:"mediumtext" json:"tags"`
    MasterPassword     string   `xorm:"varchar(100)" json:"masterPassword"`
    EnableSoftDeletion bool     `json:"enableSoftDeletion"`
    IsProfilePublic    bool     `json:"isProfilePublic"`

    AccountItems []*AccountItem `xorm:"varchar(2000)" json:"accountItems"`
}
```

## User

A user in Casdoor can log into an application. One user can only belong to one organization, but can have the ability to log into multiple applications that owned by the organization. Currently there are two types of users in Casdoor:

- Users under `built-in` organization, like `built-in/admin`: the global administrators, have the full administrator power on the Casdoor platform.
- Users under other organizations, like `my-company/alice`: normal users, can only sign up, sign in, sign out, change his/her own profile, etc.

In Casdoor API, a user is usually identified as `<organization_name>/<username>`, e.g., the default administrator of Casdoor is denoted as `built-in/admin`. There is also a property in user called `id`, which is a UUID like `d835a48f-2e88-4c1f-b907-60ac6b6c1b40`, it can also be chosen as a ID for a user by an application.

> **TIP**
> If your application is only for one organization, you can just use `<username>` instead of `<organization_name>/<username>` as user ID across your application for simplicity.

The User class definition is shown as follows:

```go
type User struct {
    Owner       string `xorm:"varchar(100) notnull pk" json:"owner"`
    Name        string `xorm:"varchar(100) notnull pk" json:"name"`
    CreatedTime string `xorm:"varchar(100)" json:"createdTime"`
    UpdatedTime string `xorm:"varchar(100)" json:"updatedTime"`

    Id                string   `xorm:"varchar(100)" json:"id"`
    Type              string   `xorm:"varchar(100)" json:"type"`
    Password          string   `xorm:"varchar(100)" json:"password"`
    PasswordSalt      string   `xorm:"varchar(100)" json:"passwordSalt"`
    DisplayName       string   `xorm:"varchar(100)" json:"displayName"`
    Avatar            string   `xorm:"varchar(500)" json:"avatar"`
    PermanentAvatar   string   `xorm:"varchar(500)" json:"permanentAvatar"`
    Email             string   `xorm:"varchar(100) index" json:"email"`
    Phone             string   `xorm:"varchar(100) index" json:"phone"`
    Location          string   `xorm:"varchar(100)" json:"location"`
    Address           []string `json:"address"`
    Affiliation       string   `xorm:"varchar(100)" json:"affiliation"`
    Title             string   `xorm:"varchar(100)" json:"title"`
    IdCardType        string   `xorm:"varchar(100)" json:"idCardType"`
    IdCard            string   `xorm:"varchar(100) index" json:"idCard"`
    Homepage          string   `xorm:"varchar(100)" json:"homepage"`
    Bio               string   `xorm:"varchar(100)" json:"bio"`
    Tag               string   `xorm:"varchar(100)" json:"tag"`
    Region            string   `xorm:"varchar(100)" json:"region"`
    Language          string   `xorm:"varchar(100)" json:"language"`
    Gender            string   `xorm:"varchar(100)" json:"gender"`
    Birthday          string   `xorm:"varchar(100)" json:"birthday"`
    Education         string   `xorm:"varchar(100)" json:"education"`
    Score             int      `json:"score"`
    Ranking           int      `json:"ranking"`
    IsDefaultAvatar   bool     `json:"isDefaultAvatar"`
    IsOnline          bool     `json:"isOnline"`
    IsAdmin           bool     `json:"isAdmin"`
    IsGlobalAdmin     bool     `json:"isGlobalAdmin"`
    IsForbidden       bool     `json:"isForbidden"`
    IsDeleted         bool     `json:"isDeleted"`
    SignupApplication string   `xorm:"varchar(100)" json:"signupApplication"`
    Hash              string   `xorm:"varchar(100)" json:"hash"`
    PreHash           string   `xorm:"varchar(100)" json:"preHash"`

    CreatedIp      string `xorm:"varchar(100)" json:"createdIp"`
    LastSigninTime string `xorm:"varchar(100)" json:"lastSigninTime"`
    LastSigninIp   string `xorm:"varchar(100)" json:"lastSigninIp"`

    Github   string `xorm:"varchar(100)" json:"github"`
    Google   string `xorm:"varchar(100)" json:"google"`
    QQ       string `xorm:"qq varchar(100)" json:"qq"`
    WeChat   string `xorm:"wechat varchar(100)" json:"wechat"`
    Facebook string `xorm:"facebook varchar(100)" json:"facebook"`
    DingTalk string `xorm:"dingtalk varchar(100)" json:"dingtalk"`
    Weibo    string `xorm:"weibo varchar(100)" json:"weibo"`
    Gitee    string `xorm:"gitee varchar(100)" json:"gitee"`
    LinkedIn string `xorm:"linkedin varchar(100)" json:"linkedin"`
    Wecom    string `xorm:"wecom varchar(100)" json:"wecom"`
    Lark     string `xorm:"lark varchar(100)" json:"lark"`
    Gitlab   string `xorm:"gitlab varchar(100)" json:"gitlab"`
    Apple    string `xorm:"apple varchar(100)" json:"apple"`
    AzureAD  string `xorm:"azuread varchar(100)" json:"azuread"`
    Slack    string `xorm:"slack varchar(100)" json:"slack"`

    Ldap       string            `xorm:"ldap varchar(100)" json:"ldap"`
    Properties map[string]string `json:"properties"`
}
```

# Application

An application represents a web service that needs to be protected by Casdoor. E.g., a forum site, an OA system, a CRM system are all applications.

```
type Application struct {
    Owner       string `xorm:"varchar(100) notnull pk" json:"owner"`
    Name        string `xorm:"varchar(100) notnull pk" json:"name"`
    CreatedTime string `xorm:"varchar(100)" json:"createdTime"`

    DisplayName         string          `xorm:"varchar(100)" json:"displayName"`
    Logo                string          `xorm:"varchar(100)" json:"logo"`
    HomepageUrl         string          `xorm:"varchar(100)" json:"homepageUrl"`
    Description         string          `xorm:"varchar(100)" json:"description"`
    Organization        string          `xorm:"varchar(100)" json:"organization"`
    Cert                string          `xorm:"varchar(100)" json:"cert"`
    EnablePassword      bool            `json:"enablePassword"`
    EnableSignUp        bool            `json:"enableSignUp"`
    EnableSigninSession bool            `json:"enableSigninSession"`
    EnableCodeSignin    bool            `json:"enableCodeSignin"`
    Providers           []*ProviderItem `xorm:"mediumtext" json:"providers"`
    SignupItems         []*SignupItem   `xorm:"varchar(1000)" json:"signupItems"`
    OrganizationObj     *Organization   `xorm:"-" json:"organizationObj"`

    ClientId             string   `xorm:"varchar(100)" json:"clientId"`
    ClientSecret         string   `xorm:"varchar(100)" json:"clientSecret"`
    RedirectUris         []string `xorm:"varchar(1000)" json:"redirectUris"`
    TokenFormat          string   `xorm:"varchar(100)" json:"tokenFormat"`
    ExpireInHours        int      `json:"expireInHours"`
    RefreshExpireInHours int      `json:"refreshExpireInHours"`
    SignupUrl            string   `xorm:"varchar(200)" json:"signupUrl"`
    SigninUrl            string   `xorm:"varchar(200)" json:"signinUrl"`
    ForgetUrl            string   `xorm:"varchar(200)" json:"forgetUrl"`
    AffiliationUrl       string   `xorm:"varchar(100)" json:"affiliationUrl"`
    TermsOfUse           string   `xorm:"varchar(100)" json:"termsOfUse"`
    SignupHtml           string   `xorm:"mediumtext" json:"signupHtml"`
    SigninHtml           string   `xorm:"mediumtext" json:"signinHtml"`
}
```

Each application can have its own customized sign up page, sign in page, etc. E.g., the root login page `/login` (like: https://door.casdoor.com/login) is the sign in page only for Casdoor's built-in application: `app-built-in`.

An application is a "portal" or "interface" for a user to log into Casdoor. A user must go through one application's sign in page to log into Casdoor.

| Application | Sign up page URL | Sign in page URL |
|---|---|---|
| app-built-in | https://door.casdoor.com/signup | https://door.casdoor.com/login |
| app-casnode | https://door.casdoor.com/signup/app-casnode | https://door.casdoor.com/login/oauth/authorize?client_id=014ae4bd048734ca2dea&response_type=code&redirect_uri=http://localhost:9000/callback&scope=read&state=casdoor |
| app-casbin-oa | https://door.casdoor.com/signup/app-casbin-oa | https://door.casdoor.com/login/oauth/authorize?client_id=0ba528121ea87b3eb54d&response_type=code&redirect_uri=http://localhost:9000/callback&scope=read&state=casdoor |

## Login URLs

It's very easy to log into Casdoor via Casdoor's built-in application, just visit Casdoor server's homepage (like: https://door.casdoor.com for demo site) and it will automatically redirect you to `/login`. But how to get these URLs for other applications in frontend and backend code? You can either concatenate strings by yourself or call some utility functions provided by Casdoor SDKs to get the URLs:

**1. By concatenating string manually:**

- Sign up page URL
  - Signup for the specified application: `<your-casdoor-hostname>/signup/<your-application-name>`
  - Signup by oauth: `<your-casdoor-hostname>/signup/oauth/authorize?client_id=<client-id-for-your-application>&response_type=code&redirect_uri=<redirect-uri-for-your-application>&&scope=read&state=casdoor`
  - Signup automatically: `<your-casdoor-hostname>/auto-signup/oauth/authorize?client_id=<client-id-for-your-application>&response_type=code&redirect_uri=<redirect-uri-for-your-application>&&scope=read&state=casdoor`
- Sign in page URL
  - Signin for the specified organization: `<your-casdoor-hostname>/login/<your-organization-name>`
  - Signin by oauth: `<your-casdoor-hostname>/login/oauth/authorize?client_id=<client-id-for-your-application>&response_type=code&redirect_uri=<redirect-uri-for-your-application>&&scope=read&state=casdoor`

**2. Use frontend SDK (for frontend Javascript code using React, Vue or Angular):**

`getSignupUrl()` and `getSigninUrl()`: casdoor-js-sdk

**3. Use backend SDK (for backend code using Go, Java, etc.):**

`GetSignupUrl()` and `GetSigninUrl()`: casdoor-go-sdk

# Provider

Casdoor is a federated single-sign-on system, which supports multiple identity providers via OIDC, OAuth and SAML. Casdoor can also send verification code or other notifications to users via Email or SMS (Short Message Service). Casdoor uses the concept: `Provider` to manage all these third-party connectors.

Currently, All providers supported by Casdoor can be found here: provider/overview

```go
type Provider struct {
    Owner       string `xorm:"varchar(100) notnull pk" json:"owner"`
    Name        string `xorm:"varchar(100) notnull pk" json:"name"`
    CreatedTime string `xorm:"varchar(100)" json:"createdTime"`

    DisplayName   string `xorm:"varchar(100)" json:"displayName"`
    Category      string `xorm:"varchar(100)" json:"category"`
    Type          string `xorm:"varchar(100)" json:"type"`
    Method        string `xorm:"varchar(100)" json:"method"`
    ClientId      string `xorm:"varchar(100)" json:"clientId"`
    ClientSecret  string `xorm:"varchar(100)" json:"clientSecret"`
    ClientId2     string `xorm:"varchar(100)" json:"clientId2"`
    ClientSecret2 string `xorm:"varchar(100)" json:"clientSecret2"`

    Host    string `xorm:"varchar(100)" json:"host"`
    Port    int    `json:"port"`
    Title   string `xorm:"varchar(100)" json:"title"`
    Content string `xorm:"varchar(1000)" json:"content"`

    RegionId     string `xorm:"varchar(100)" json:"regionId"`
    SignName     string `xorm:"varchar(100)" json:"signName"`
    TemplateCode string `xorm:"varchar(100)" json:"templateCode"`
    AppId        string `xorm:"varchar(100)" json:"appId"`
```

# How does Casdoor manage itself?

When you run Casdoor for the first time, Casdoor will create some built-in objects to help the administrator to manage Casdoor itself:

- A built-in organization named `built-in`.
- A user named `admin` in the `built-in` organization.
- A built-in application named `app-built-in`, owned by the `built-in` organization, representing Casdoor itself (Casdoor is actually also an application).

All the users under `built-in` organization, including `admin` will have the full administrator power on the Casdoor platform by default. So if you have multiple administrators, then create new accounts under `built-in` organization. Otherwise, remember to close the sign up channel for the `app-built-in` application.

> ⚠ CAUTION
>
> The built-in objects are already forbidden to rename or delete in both web UI or RESTful API. Casdoor has hard-coded these reserved names in many places. Do not try to rename or delete them in any way like modifying the DB, otherwise the whole system may crash.

# Server Installation

## Requirements

### OS

All major operating systems including Windows, Linux and macOS are supported.

### Environment

- Go 1.17+
- Node.js LTS (16 or 14)
- Yarn 1.x

> INFO
>
> We strongly suggest you use Yarn 1.x to run & build Casdoor frontend, using NPM might cause UI styling issues, see more details at: casdoor#294

> CAUTION
>
> For **Chinese** users, in order to download the Go dependency packages successfully, you need to use a Go proxy by Configuring the GOPROXY environment variable. We strongly recommend: https://goproxy.cn/

### Database

Casdoor uses XORM to talk to the database. Based on Xorm Drivers Support,

Casdoor currently provides support for following databases:

- `MySQL`
- `MariaDB`
- `PostgreSQL`
- `CockroachDB`
- `SQL Server`
- `Oracle`
- `SQLite 3`
- `TiDB`

# Download

The source code of Casdoor is hosted at GitHub: https://github.com/casdoor/casdoor. Both the Go backend code and React frontend code are inside the single repository.

| Name | Description | Language | Source code |
|---|---|---|---|
| Frontend | Web frontend UI for Casdoor | JavaScript + React | https://github.com/casdoor/casdoor/tree/master/web |
| Backend | RESTful API backend for Casdoor | Golang + Beego + XORM | https://github.com/casdoor/casdoor |

Casdoor supports `Go Modules`. To download the code, you can just simply clone the code via git:

```
cd path/to/folder
git clone https://github.com/casdoor/casdoor
```

# Configuration

## Configure Database

Casdoor supports mysql, mssql, sqlite3, postgres. Casdoor uses mysql by default. If you use another database, please modify the import package in object/adapter.

MySQL:

Casdoor will store its users, nodes and topics information in a MySQL database named: `casdoor`. If the database does not exist, it needs to be created manually. The DB connection string can be specified at: https://github.com/casdoor/casdoor/blob/master/conf/app.conf

```
driverName = mysql
dataSourceName = root:123456@tcp(localhost:3306)/
dbName = casdoor
```

PostgreSQL:

Since we must choose a database when opening Postgres with xorm, you should prepare a database manually before running Casdoor.

Let's assume that you have already prepared a database called `casdoor`, then you should specify `app.conf` like this:

```
driverName = postgres
```

> INFO
>
> For PostgreSQL, make sure `dataSourceName` has non-empty `dbName` and leave the standalone `dbName` field empty like the above example.

CockroachDB:

You can also use cockroachdb with postgres driver. It has same configuration as postgreSQL.

```
driverName = postgres
dataSourceName = "user=postgres password=postgres host=localhost
port=5432 sslmode=disable dbname=casdoor
serial_normalization=virtual_sequence"
dbName =
```

> INFO
>
> For CockroachDB, don't forget to add `serial_normalization=virtual_sequence` to the `dataSourceName` like the above example. otherwise you will get error regarding existed database, whenever the service started or restarted. Notice, this must be added before the database created.

Sqlite3

You should specify `app.conf` like this:

```
driverName = sqlite
dataSourceName = "file:casdoor.db?cache=shared"
dbName = casdoor
```

## Via Ini file

Casdoor can be configured via a single file: conf/app.conf, the content of which by default is:

```
appname = casdoor
httpport = 8000
runmode = dev
SessionOn = true
copyrequestbody = true
driverName = mysql
dataSourceName = root:123456@tcp(localhost:3306)/
dbName = casdoor
tableNamePrefix =
showSql = false
redisEndpoint =
defaultStorageProvider =
isCloudIntranet = false
authState = "casdoor"
socks5Proxy = "127.0.0.1:10808"
verificationCodeTimeout = 10
initScore = 2000
logPostOnly = true
origin = "https://door.casdoor.com"
staticBaseUrl = "https://cdn.casbin.org"
enableGzip = true
```

- `appname` is the application name, which currently has no practical use
- `httpport` is the port that your back-end application is listening on
- `runmode` is `dev` or `prod`
- `SessionOn` decides whether to enable session,used by default.
- `driverName`, `dataSourceName` and `dbName` are introduced before, please see Configure Database.

- `verificationCodeTimeout` set the expiration time of the verification code. After the expiration, the user needs to obtain it again

Despite all the configurable fields, as a beginner, you only need to modify two items: `driverName` and `dataSourceName` based on your database. This database will be used by Casdoor to store all data, including users, organizations, applications and so on.

- `tableNamePrefix` is prefix of the table when using adapter.
- `showSql` : show SQL statement or not on logger if log level is great than INFO.
- `redisEndpoint` is the Redis endpoint used by Beego session storage. If this parameter is empty, the session data will be stored locally as files in `./tmp` folder. For using Redis as Beego session storage, an example for this value would be: `redis.example.com:6379`. If Redis is deployed in the local machine, you can use `localhost:6379`. If Redis password is enabled, use `redis.example.com:6379,db,password`. See more details at: https://beego.run/docs/module/session.md#saving-provider-config
- `defaultStorageProvider` is the default file storage service name. If you need to use file storage services such as avatar upload, you need to set up a storage provider and apply it in your application. See storage for details.
- `isCloudIntranet` is used to identify whether your provider endpoint is intranet endpoint.
- `authState` is the authorization application name. This parameter will be checked when logging in.
- `socks5Proxy` is the SOCKS proxy server IP address. Set the proxy port, because we have google-related services or use `Google` `GitHub` `Facebook` `LinkedIn` `Steam` as OAuth Provider which will be restricted by the network in some areas.
- `initScore` is the initial score of each user. Each user has a score attribute. Score is used by Casnode. Score does not control anything in Casdoor.

- `logPostOnly` is used to identify whether only use post method to add a record.
- `origin` is the origin backend domain name.
- `staticBaseUrl` is address of the static image when the system initializes the database.
- `enableGzip` will accept and response with gzip encoding when the request header including `Accept-Encoding=gzip`.

## Via Environment Variables

**All configuration items defined by Casdoor** in the ini file mentioned above can be chosen to configuration via environmental variables, so can **some of the beego configurations** items(httpport,appname).

For example, when you try to start casdoor, you can use something like this to pass the configuration via environmental variables.

```
appname=casbin go run main.go
```

Besides, `export` derivatives are also a possible method. The names of environmental variables should be exactly the same with the names you want to use in the ini file.

*Note: configurations in environmental variables can **override** the configurations in ini file.*

# Run

There are currently two methods to start, you can choose one according to your own situation.

# Development mode

### Backend

Casdoor's Go backend runs at port 8000 by default. You can start the Go backend with the following command:

```
go run main.go
```

After the server is successfully running, we can start the frontend part.

### Frontend

Casdoor's frontend is a very classic Create-React-App (CRA) project. It runs at port `7001` by default. Use the following commands to run the frontend:

```
cd web
yarn install
yarn start
```

Visit: http://localhost:7001 in your browser. Log into Casdoor dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

## Production mode

### Backend

Build Casdoor Go backend code into executable and start it.

For Linux:

```
go build
./casdoor
```

For Windows:

```
go build
casdoor.exe
```

### Frontend

Build Casdoor frontend code into static resources (.html, .js, .css files):

```
cd web
yarn install
yarn build
```

Visit: http://localhost:8000 in your browser. Log into Casdoor dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

TIP

To use another port, please edit `conf/app.conf` and modify `httpport`, then restart the Go backend.

CASDOOR PORT DETAILS

In **dev** environment, the frontend is run by `yarn run` in port 7001, so if you want to go to Casdoor login page, you need set Casdoor link as **http://localhost:7001**.

In **prod** environment, the frontend files is first built by `yarn build` and served in port 8000, so if you want to go to Casdoor login page, you need to set Casdoor link as **http://SERVER_IP:8000** (If you are using reverse proxy, you need to set the link as **your domain**).

**Take our official forum Casnode as an example:**

Casnode uses Casdoor to handle authentication.

When we are testing Casnode in **dev** environment, we set the `serverUrl` as http://localhost:7001, so when we test signin and signup functionality using Casdoor, it will go to localhost 7001 which is the Casdoor port.

And when we put Casnode to **prod** environment, we set the `serverUrl` as https://door.casdoor.com, so users can signin or signup using Casdoor.

```
import * as ConfBackend from "./backend/ConfBackend.js"

export const AuthConfig = {
  // serverUrl: "https://door.casbin.com",
  serverUrl: "http://localhost:7001",
  clientId: "014ae4bd048734ca2dea",
```

# (Optional) Try with Docker

## Requirements

### Hardware

If you want to build the Docker image by yourself, please ensure that your machine has at least **2GB** memory. Casdoor's frontend is a NPM project of React. Building the frontend requires at least **2GB** memory. Less than **2GB** memory may lead to frontend build failure.

If you just need to run the pre-built image, please ensure that your machine has at least **100MB** memory.

### OS

All OSes (Linux, Windows and macOS) are supported.

### Docker

You can use **docker (docker-engine version >= 17.05)** in Linux or **Docker Desktop** in Windows and macOS.

- Docker

Users of all OSes must ensure that the **docker-engine version >= 17.05**. It is because we use multi-stage build feature in docker-compose.yml, which was supported in 17.05 and above versions. See https://docs.docker.com/develop/develop-images/multistage-build/ for more information.

If you also use docker-compose, please ensure that **docker-compose version >= 2.2**. For Linux users, you also need to make sure that docker-compose is installed, given that it is separated from docker-engine.

# Get the image

We have provided two DockerHub images:

| Name | Description | Suggestion |
| --- | --- | --- |
| casdoor-all-in-one | Both Casdoor and a MySQL database are inside the image | Already includes a toy database and only for test purpose |
| casdoor | Only Casdoor is inside the image | Can be connected to your own database and used in production |

1. casbin/casdoor-all-in-one, in which casdoor binary, a mysql database and all necessary configurations are packed up. This image is for new users to have a trial on casdoor quickly. **With this image you can start a casdoor immediately with one single command (or two) without any complex configuration. Note: we DO NOT recommend you to use this image in productive environment**

## Option-1: Use the toy database

Run the container with port `8000` exposed to host. It will automatically pull the image if it doesn't exist in the local host.

```
docker run -p 8000:8000 casbin/casdoor-all-in-one
```

> ⚠ CAUTION
>
> Some users in areas like China usually use Docker image mirror services like Alibaba Cloud Image Booster (English) to achieve higher download speed compared to DockerHub. However, it has a known issue that the `latest` tag provided by those services is not up-to-date. It probably results in a very old image by fetching the `latest` tag. To mitigate this issue, you can specify the image version number explicitly by using the following command:
>
> ```
> docker pull casbin/casdoor-all-in-one:$(curl -sS "https://hub.docker.com/v2/repositories/casbin/casdoor-all-in-one/tags/?page_size=1&page=2" | sed 's/,/,\n/g' | grep '"name"' |awk -F '"' '{print $4}')
> ```
>
> Note: the above command utilizes Linux tools like `curl`, `sed`, `grep`, `awk`. If you are using Windows, make sure you run it in a Linux-style shell like `Git Shell` or `Cygwin`. `CMD` or `PowerShell` won't work.

Visit: http://localhost:8000 in your browser. Log into Casdoor dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

## Option-2: Try with docker-compose

> ⚠ CAUTION
>
> Some users in areas like China usually use Docker image mirror services

> like Alibaba Cloud Image Booster (English) to achieve higher download speed compared to DockerHub. However, it has a known issue that the `latest` tag provided by those services is not up-to-date. It probably results in a very old image by fetching the `latest` tag. To mitigate this issue, you can specify the image version number explicitly by using the following command:
>
> ```
> docker pull casbin/casdoor:$(curl -sS "https://hub.docker.com/v2/repositories/casbin/casdoor/tags/?page_size=1&page=2" | sed 's/,/,\n/g' | grep '"name"' |awk -F '"' '{print $4}')
> ```
>
> Note: the above command utilizes Linux tools like `curl`, `sed`, `grep`, `awk`. If you are using Windows, make sure you run it in a Linux-style shell like `Git Shell` or `Cygwin`. `CMD` or `PowerShell` won't work.

Create a `conf/app.conf` directory in the same level directory of the `docker-compose.yml` file, then copy app.conf from Casdoor. For more details about `app.conf`, you can see Via Ini file.

Create a separate database by docker-compose:

```
docker-compose up
```

That's it! 🛩️

Visit: http://localhost:8000 in your browser. Log into Casdoor dashboard with the default global admin account: `built-in/admin`

```
admin
```

*Note: if you dive deeper into the docker-compose.yml, you may be puzzled by the environment variable we created in it called "RUNNING_IN_DOCKER". When database 'db' is created via docker-compose, it is available on localhost of your pc but not localhost of the casdoor container. To prevent you from the troubles caused by modifying app.conf which are pretty difficult for a new user, we provided this environment variable and pre-assigned it in docker-compose.yml. When this environment variable is true, localhost will be replaced with host.docker.internal so that you casdoor can visit the db.*

## Option-3 Try directly with standard image

> ⚠ CAUTION
>
> Some users in areas like China usually use Docker image mirror services like Alibaba Cloud Image Booster (English) to achieve higher download speed compared to DockerHub. However, it has a known issue that the `latest` tag provided by those services is not up-to-date. It probably results in a very old image by fetching the `latest` tag. To mitigate this issue, you can specify the image version number explicitly by using the following command:
>
> ```
> docker pull casbin/casdoor:$(curl -sS "https://hub.docker.com/v2/repositories/casbin/casdoor/tags/?page_size=1&page=2" | sed 's/,/,\n/g' | grep '"name"' |awk -F '"' '{print $4}')
> ```
>
> Note: the above command utilizes Linux tools like `curl`, `sed`, `grep`, `awk`. If you are using Windows, make sure you run it in a Linux-style shell like `Git Shell` or `Cygwin`. `CMD` or `PowerShell` won't work.

TIP

if it is not convenient to mount the configuration file to a container, using environment variables is also a possible solution.

example

```
docker run \
  -e driverName=mysql \
  -e dataSourceName='user:password@tcp(x.x.x.x:3306)/' \
  -p 8000:8000 \
  casbin/casdoor:latest
```

Create `conf/app.conf`, you can copy it from conf/app.conf in Casdoor. For more details about `app.conf`, you can see Via Ini file.

Then run

```
docker run  -p 8000:8000 -v /folder/of/app.conf:/conf casbin/
casdoor:latest
```

Anyway just **mount the app.conf to /conf/app.conf** and start it.

Visit: http://localhost:8000 in your browser. Log into Casdoor dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

# Casdoor Public API

Casdoor is developed in a frontend and backend separated manner (in contrast to JSP or PHP). The Go backend only exposes its functionality via RESTful API. The React frontend code consumes the RESTful API to render the web UI and perform actions. We call the RESTful API as `Casdoor Public API`. The API can usually be used by:

- Casdoor's frontend
- Casdoor client SDKs
- Any other customized code from the application side

The full reference of `Casdoor Public API` can be viewed at Swagger: https://door.casdoor.com/swagger . This Swagger docs are automatically generated by Beego's Bee tool.

# Tutorials

## Product documentation

| Product | Technologies | Docs |
| --- | --- | --- |
| Dashboard of PingCAP TiDB | React + Typescript + Go + Gin | Use Casdoor for TiDB Dashboard SSO sign-in (other languages: Chinese, Japanese) |
| GitLab | Vue + Ruby + Rails | OpenID Connect OmniAuth provider |
| Apache Shenyu | Java | Casdoor Plugin (other languages: Chinese) |
| Alist | Typescript + SoildJS + Go + Gin | Casdoor SSO (other languages: Chinese) |
| BookStack | jQuery + Bootstrap + Go + Beego | Casdoor integrates registration and login |

# Articles

| Technologies | Language | Title |
| --- | --- | --- |
| ASP.Net Core 6 | English | ASP.Net Core .net 6 Demo Authentication Project using local Casdoor Docker Container on Windows Subsystem for Linux |
| OAuth2 Proxy (Go) | Chinese | Use Casdoor + OAuth-Proxy to protect web applications on public networks |
| Casnode (Javascript + React + Go + Beego) | Chinese | Use Lighthouse to set up a forum like v2ex |
| Cloudreve (Go) | Chinese | Modify Cloudreve to support Casdoor |
| KodExplorer (PHP) | Chinese | Modify KodExplorer to support Casdoor |

# Deployment

## Data Initialization

How to initialize Casdoor data from files

## Hosting Static Files in CDN

Hosting frontend static files in the CDN

## Hosting Static Files in Intranet

How to deploy Casdoor static resource

## DB Migration

Handle DB Migration in Casdoor

# Data Initialization

If you are deploying Casdoor with other services as a whole application, you may want to provide an **out-of-box** feature for users (User can directly use the application without any configuration).

For such a situation, you can use data initialization to register your service in Casdoor through one configuration file. This file can be pre-defined or dynamically generated by your owner service.

## How to use

If there is one configuration file named as `init_data.json` at the root directory of Casdoor, it will be used to initialize data in Casdoor. **What you should do is just to put this file at the root directory where Casdoor will run.**

If you use official docker image of Casdoor, the following are some scripts that can help you to mount `init_data.json` into the container.

### Docker

If you deploy Casdoor with docker, you can use the `volume` to mount `init_data.json` into the container.

```
docker run ... -v /path/to/init_data.json:/init_data.json
```

## Kubernetes

If you deploy Casdoor with kubernetes, you can use the `configmap` to store `init_data.json`.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: casdoor-init-data
data:
  init_data.json:
```

You can mount the data into Casdoor `pods` by mounting the `configmap`. You may modify your `deployment` as follows:

```yaml
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    ...
    spec:
      containers:
      ...
        volumeMounts:
        - mountPath: /init_data.json
          name: casdoor-init-data-volume
          subPath: init_data.json
      volumes:
      - configMap:
          name: casdoor-init-data
        name: casdoor-init-data-volume
```

# File details

There is already a template named `init_data.json.template` at the root directory of Casdoor repository. You can refer to this file to customize your initialization.

The following is the Go struct of each part mapping to and their documentation:

| Object | Go Struct | Documentation |
|---|---|---|
| organizations | stuct | doc |
| applications | stuct | doc |
| users | stuct | doc |
| providers | stuct | doc |
| certs | stuct | |
| ldaps | stuct | doc |

If you still feel confused of filling this template, you can call restful api or use the debug mode of your browser to see the response of `GetXXX` to these objects. These response are in the same format as `init_data.json`.

# Hosting Static Files in CDN

Frontend static resource (.js, .css files) are in `web/build/static/`. If you want to deploy it in a **CDN** service of a public cloud, Casdoor provides a script for you to deploy frontend static files easily. Please follow the steps below.

> NOTE
>
> We assume you have already built the frontend code of Casdoor. If not yet, please follow: document.

## Preparation

First, you need to create a valid Storage Provider in Casdoor UI. you can refer to the example.

> CAUTION
>
> When you fill in the field `Domain`, please end with '/'
>
> Domain : https://cdn.casbin.com/casdoor/

## Usage

The script is at deployment/deploy_test.go.

In deploy_test.go, you need to modify the parameter `id` in `GetProvider()`. The format of provider `id` is `<owner>/<name>`

```
func TestDeployStaticFiles(t *testing.T) {
    provider := object.GetProvider("admin/
provider_storage_aliyun_oss")
    deployStaticFiles(provider)
}
```

Then use the following commands to run the script:

```
cd deployment
go test
```

If the execution succeeds, you will see:

```
PASS
ok      github.com/casdoor/casdoor/deployment   2.951s
```

# How it works

The script will:

- It will upload all the files in folders: `css/` and `js/` to the CDN service pointed by the storage provider.
- Replace all the URLs of `.css` and `.js` files in the `web/build/index.html` with the URLs hosted in the CDN.

You still need to keep the `index.html`. After the static files are uploaded to CDN, `index.html` will still be requested by users through Casdoor Go backend, and those static files in the CDN are then requested through the URLs in `index.html`.

# Hosting Static Files in Intranet

If you are deploying Casdoor on an **Intranet**, you may not be able to access the static resource directly over the Internet. You need to deploy static resources where you can access them, and then modify the configuration in Casdoor in 3 places.

## Deploy static resource

All static resources in Casdoor, including images, logo, css, etc., are stored in casbin/static repository.

**Clone** the repository and **deploy** it on a web servers. Make sure you can access the resource.

## Modify in Casdoor

You can simply modify the configuration file to set the static resource address to where you deployed it. Go to conf/app.conf, set `staticBaseUrl` to your deployed address.

```
staticBaseUrl = "https://cdn.casbin.org"
```

# DB Migration

When the database is upgraded, it is easy to have a data crash, for example, we need to delete an old field. Fortunately, the xorm used by Casdoor will help us with a lot of database migration problems. But we still need to handle some schema and data migrations ourselves, such as a **field name changed**

> NOTE
>
> You can understand xorm Schema operation in xorm docs

## How it works

As mentioned above, when a field name changes, xorm will not be able to do anything, but xorm provides a migrate package to help us solve this problem.

You can write code like this to handle field renaming:

```
migrations := []*migrate.Migration{
        {
            ID: "CasbinRule--fill ptype field with p",
            Migrate: func(tx *xorm.Engine) error {
                _, err :=
tx.Cols("ptype").Update(&xormadapter.CasbinRule{
                    Ptype: "p",
                })
                return err
            },
            Rollback: func(tx *xorm.Engine) error {
```

What we want to achieve is: **rename `p_type` to `ptype`**. But since xorm **does not support field renaming**, we can only use a more complicated way: assign the value of `p_type` to `ptype`, and then delete the `p_type` field.

The `ID` field uniquely refers to the migration we performed. After the `m.Migrate()` runs, and the value of the `ID` will be added to the migrations table of the database.

When the project is started again, the database will check the existing `ID` field in the table and will not perform operations with the same `ID`

# How to Connect to Casdoor


## Overview

Connect your app to Casdoor

## Standard OIDC Client

Using OIDC discovery to migrate to Casdoor

## Casdoor SDKs

Using Casdoor SDKs instead of standard OIDC protocol

## How to Enable Single Sign-On

Enable Single Sign-On

## Vue SDK

Casdoor Vue SDK

## 🗃️ Desktop SDKs

3 items

## 📄 Casdoor Plugin

Using Casdoor plugins or middlewares in other frameworks like Spring Boot, WordPress, Odoo, etc.

## 📄 OAuth 2.0

Using AccessToken to authenticate clients

## 📄 CAS

Using Casdoor as CAS server

## 🗃️ SAML

3 items

## 📄 WebAuthn

Use WebAuthn in Casdoor

# Overview

In this section, we will show how to connect your application to Casdoor.

As Service Provider (SP), Casdoor supports two authentication protocols:

- `OAuth 2.0 (OIDC)`
- `SAML`

As Identity Provider (IdP), Casdoor supports 4 authentication protocols:

- `OAuth 2.0`
- `OIDC`
- `SAML`
- `CAS 1.0, 2.0, 3.0`

## OAuth 2.0 (OIDC)

What is OAuth 2.0?

OAuth 2 is an authorization framework that enables applications — such as Facebook, GitHub, and Casdoor — to obtain limited access to user accounts on an HTTP service. It works by delegating user authentication to the service that hosts a user account and authorizing third-party applications to access that user account. OAuth 2 provides authorization flows for web and desktop applications, as well as mobile devices.

Casdoor's authorization process is built upon the OAuth 2.0 protocol. We recommend using the OAuth 2.0 protocol:

1. The protocol is simple and easy to implement, and can solve many scenarios.
2. High maturity and extensive community support

Therefore, your application will talk to Casdoor via OAuth 2.0 (OIDC). Specifically, there are three ways for connecting to Casdoor:

## Standard OIDC client

Standard OIDC client: use a standard OIDC client implementation, which is usually widely provided in any programming language or framework.

What is OIDC?

OpenID Connect (OIDC) is an open authentication protocol that works on top of the OAuth 2.0 framework. Targeted toward consumers, OIDC allows individuals to use single sign-on (SSO) to access relying party sites using OpenID Providers (OPs), such as an email provider or social network, to authenticate their identities. It provides the application or service with information about the user, the context of their authentication, and access to their profile information.

Casdoor has fulfilled the OIDC protocol completely. If your application is already running against another OAuth 2.0 (OIDC) identity provider via a **standard OIDC client library**, and you want to migrate to Casdoor, using OIDC discovery will be very **easy** for you to switch to Casdoor.

## Casdoor SDKs

Casdoor SDKs: For most programming languages, Casdoor will provide easy-to-use SDK library on top of OIDC, with supporting extended functionality which are only available in Casdoor.

Compared to the standard OIDC protocol, Casdoor provides more functionalities in its SDK, like user management, resource uploading, etc. Connecting to Casdoor via Casdoor SDK costs more time than using a standard OIDC client library but will provide the best **flexibility** and the most **powerful** API.

## Casdoor plugin

Casdoor plugin: if your application is built on top of a popular platform (like Spring Boot, WordPress, etc.) and Casdoor (or a third-party) has already provided a plugin or middleware for it, then use it. It will be much easier to use a plugin than manually invoking Casdoor SDK because the former is specially made for the platform.

**plugin:**

- Jenkins plugin
- APISIX plugin

**Middleware:**

- Spring Boot plugin
- Django plugin

# SAML

What is SAML?

Security Assertion Markup Language (SAML) is an open standard that allows identity providers (IdP) to pass authorization credentials to service providers (SP). What that jargon means is that you can use one set of credentials to log into many different websites. It's much simpler to manage one login per user than it is to manage separate logins to email, customer relationship management (CRM) software, Active Directory, etc.

SAML transactions use Extensible Markup Language (XML) for standardized communications between the identity provider and service providers. SAML is the link between the authentication of a user's identity and the authorization to use a service.

Casdoor can be used as **SAML IdP**. Up to now the Casdoor has supported the main features of **SAML 2.0**. More details see SAML.

Example:

Casdoor as a SAML IdP in keycloak

Suggestion:

1. The protocol is **powerful** and covers many scenarios, which can be said to be one of the most comprehensive SSO protocols.
2. The protocol is **too large**, and there are many optional parameters, so it is difficult to cover all application scenarios 100% in the actual implementation.
3. If the application is **newly** developed, SAML is **not** recommended because of

its high technical complexity.

## CAS

What is CAS?

The Central Authentication Service (CAS) is a single sign-on protocol for the web. Its purpose is to permit a user to access multiple applications while providing their credentials (such as user ID and password) only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password.

Casdoor has implemented CAS 1.0, 2.0, 3.0 features. More details see CAS.

Suggestion:

1. The protocol itself is relatively lightweight and easy to implement, but it can solve a single scenario.
2. The mutual trust between the CAS Client and the CAS Server is established through interface invocation without any encryption or signature mechanism to ensure further security.
3. CAS protocol has no advantage over other protocols.

# Integrations table

Some applications already have examples that connect to Casdoor. You can follow the documentation to quickly connect to Casdoor. You can see all applications in Integrations table.

# Standard OIDC Client

## OIDC discovery

Casdoor has fulfilled the OIDC protocol completely. If your application is already running against another OAuth 2.0 (OIDC) identity provider via a standard OIDC client library and you want to migrate to Casdoor, using OIDC discovery will be very easy for you to switch to Casdoor. Casdoor's OIDC discovery URL is:

```
<your-casdoor-backend-host>/.well-known/openid-configuration
```

E.g., the OIDC discovery URL for the demo site is: https://door.casdoor.com/.well-known/openid-configuration , with the following content:

```
{
  "issuer": "https://door.casdoor.com",
  "authorization_endpoint": "https://door.casdoor.com/login/oauth/
authorize",
  "token_endpoint": "https://door.casdoor.com/api/login/oauth/
access_token",
  "userinfo_endpoint": "https://door.casdoor.com/api/userinfo",
  "jwks_uri": "https://door.casdoor.com/.well-known/jwks",
  "introspection_endpoint": "https://door.casdoor.com/api/login/
oauth/introspect",
  "response_types_supported": [
    "code",
    "token",
    "id_token",
    "code token",
    "code id_token",
```

# List of OIDC client libraries

Here we list a few OIDC client libraries for some languages like Go and Java:

| OIDC client library | Language | Link |
|---|---|---|
| go-oidc | Go | https://github.com/coreos/go-oidc |
| pac4j-oidc | Java | https://www.pac4j.org/docs/clients/openid-connect.html |

The above table is far from being complete. For a full list of OIDC client libraries, please see more details at:

1. https://oauth.net/code/
2. https://openid.net/
    i. Certified OpenID Connect Implementations
    ii. Uncertified OpenID Connect Implementations

# OIDC UserInfo fields

The following table shows how OIDC UserInfo fields (via `/api/userinfo` API) are mapped from properties of Casdoor's User table:

| Casdoor User Field | OIDC UserInfo Field |
| --- | --- |
| Id | sub |
| originBackend | iss |
| Aud | aud |
| Name | preferred_username |
| DisplayName | name |
| Email | email |
| Avatar | picture |
| Location | address |
| Phone | phone |

See UserInfo's definition here: https://github.com/casdoor/casdoor/blob/95ab2472ce84c479be43d6fc4db6533fc738b259/object/user.go#L175-L185

# Casdoor SDKs

## Introduction

Compared to the standard OIDC protocol, Casdoor provides more functionalities in its SDK, like user management, resource uploading, etc. Connecting to Casdoor via Casdoor SDK costs more time than using a standard OIDC client library but will provide the best flexibility and the most powerful API.

Casdoor SDKs can be divided into two categories:

1. **Frontend SDK**: Like Javascript SDK, Vue SDK for websites, Android or iOS SDKs for Apps, etc. Casdoor supports providing authentication for both websites and mobile Apps.
2. **Backend SDK**: SDKs for backend languages like Go, Java, Node.js, Python, PHP, etc.

> **TIP**
> If your website is developed in a frontend and backend separated manner, then you can use the Javascript SDK: `casdoor-js-sdk` or React SDK: `casdoor-react-sdk` or Vue SDK: `casdoor-vue-sdk` to integrate Casdoor in frontend. If your web application is a traditional website developed by JSP or PHP, you can just use the backend SDKs only. See an example: casdoor-python-vue-sdk-example

| Frontend SDK | Description | SDK code | Example code |
|---|---|---|---|
| Javascript SDK | For websites | casdoor-js-sdk | casdoor-raw-js-example, Casnode, Casbin-OA, Confita |
| Android SDK | For Android apps | casdoor-android-sdk | casdoor-android-example |
| iOS SDK | For iOS apps | casdoor-ios-sdk | casdoor-ios-example |
| Electron SDK | For Electron apps | casdoor-js-sdk | casdoor-electron-example |
| .NET Desktop SDK | For .NET desktop apps | casdoor-dotnet-sdk | WPF: casdoor-dotnet-desktop-example, WinForms: casdoor-dotnet-winform-example |
| React SDK | For React websites | casdoor-react-sdk | casdoor-nodejs-react-example, casdoor-spring-security-react-example |
| Vue SDK | For Vue websites | casdoor-vue-sdk | casdoor-python-vue-sdk-example |
| Angular SDK | For Angular 1.0, 2.0 websites | casdoor-angular-sdk | |
| Flutter SDK | For Flutter apps | casdoor-flutter-sdk | casdoor-flutter-example |
| uni-app SDK | For uni-app apps | casdoor-uniapp-sdk | casdoor-uniapp-example |

Next, use one of the following backend SDKs based on the language of your backend:

| Backend SDK | Description | Sdk code | Example code |
|---|---|---|---|
| Go SDK | For Go backends | casdoor-go-sdk | Casnode, Casbin-OA, Confita |
| Java SDK | For Java backends | casdoor-java-sdk | casdoor-spring-boot-starter, casdoor-spring-boot-example, casdoor-spring-security-react-example |
| Node.js SDK | For Node.js backends | casdoor-nodejs-sdk | casdoor-nodejs-react-example |
| Python SDK | For Python backends | casdoor-python-sdk | casdoor-python-vue-sdk-example, casdoor-python-js-sdk-example, casdoor-fastapi-js-sdk-example |
| PHP SDK | For PHP backends | casdoor-php-sdk | wordpress-casdoor-plugin |
| .NET SDK | For ASP.NET backends | casdoor-dotnet-sdk | casdoor-dotnet-sdk-example |
| Rust SDK | For Rust backends | casdoor-rust-sdk | casdoor-rust-example |
| C/C++ SDK | For C/C++ backends | casdoor-cpp-sdk | casdoor-cpp-qt-example |
| Dart SDK | For Dart backends | casdoor-dart-sdk | |

For a full list of the official Casdoor SDKs, please see: https://github.com/orgs/casdoor/repositories?q=sdk&type=all&language=&sort=

# How to use Casdoor SDK?

## 1. Backend SDK configuration

When your application starts up, you need to initialize the Casdoor SDK config by calling the `InitConfig()` function with required parameters. Take casdoor-go-sdk as example: https://github.com/casbin/casnode/blob/6d4c55f5c9a3c4bd8c85f2493abad3553b9c7ac0/controllers/account.go#L51-L64

```
var CasdoorEndpoint = "https://door.casdoor.com"
var ClientId = "541738959670d221d59d"
var ClientSecret = "66863369a64a5863827cf949bab70ed560ba24bf"
var CasdoorOrganization = "casbin"
var CasdoorApplication = "app-casnode"

//go:embed token_jwt_key.pem
var JwtPublicKey string

func init() {
    auth.InitConfig(CasdoorEndpoint, ClientId, ClientSecret, JwtPublicKey, CasdoorOrganization, CasdoorApplication)
}
```

All the parameters for `InitConfig()` are explained as follows:

| Parameter | Must | Description |
|---|---|---|
| endpoint | Yes | Casdoor Server URL, like `https://door.casdoor.com` or `http://localhost:8000` |
| clientId | Yes | Client ID for the Casdoor application |
| clientSecret | Yes | Client secret for the Casdoor application |
| jwtPublicKey | Yes | The public key for the Casdoor application's cert |
| organizationName | Yes | The name for the Casdoor organization |
| applicationName | No | The name for the Casdoor application |

**TIP**

The `jwtPublicKey` can be managed in the `Certs` page as below.

You can find the public key in the cert edit page, copy it or download it for the sdk.

Then you can select the cert in the application edit page.

## 2. Frontend configuration

First, install `casdoor-js-sdk` via NPM or Yarn:

```
npm install casdoor-js-sdk
```

Or:

```
yarn add casdoor-js-sdk
```

Then define the following utility functions (better in a global JS file like `Setting.js`):

```
import Sdk from "casdoor-js-sdk";

export function initCasdoorSdk(config) {
  CasdoorSdk = new Sdk(config);
}

export function getSignupUrl() {
  return CasdoorSdk.getSignupUrl();
}

export function getSigninUrl() {
  return CasdoorSdk.getSigninUrl();
}

export function getUserProfileUrl(userName, account) {
  return CasdoorSdk.getUserProfileUrl(userName, account);
}

export function getMyProfileUrl(account) {
  return CasdoorSdk.getMyProfileUrl(account);
}

export function getMyResourcesUrl(account) {
  return CasdoorSdk.getMyProfileUrl(account).replace("/account?", "/resources?");
}
```

In the entrance file of your frontend code (like `index.js` or `app.js` in React), you need to initialize the `casdoor-js-sdk` by calling the `InitConfig()` function with required parameters. The first 4 parameters should use the same value as the Casdoor backend SDK. The last parameter `redirectPath` is relative path for the redirected URL, returned from Casdoor's login page.

```
const config = {
  serverUrl: "https://door.casdoor.com",
  clientId: "014ae4bd048734ca2dea",
  organizationName: "casbin",
  appName: "app-casnode",
  redirectPath: "/callback",
};

xxx.initCasdoorSdk(config);
```

**(Optional)** Because we are using React as example, our `/callback` path is hitting the React route. We use the following React component to receive the `/callback` call and send to the backend. You can ignore this step if you are redirecting to backend directly (like in JSP or PHP).

```
import React from "react";
import {Button, Result, Spin} from "antd";
import {withRouter} from "react-router-dom";
import * as Setting from "./Setting";

class AuthCallback extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      classes: props,
      msg: null,
    };
  }

  componentWillMount() {
    this.login();
  }

  login() {
    Setting.signin().then((res) => {
      if (res.status === "ok") {
        Setting.showMessage("success", `Logged in successfully`);
        Setting.goToLink("/");
      } else {
        this.setState({
          msg: res.msg,
        });
      }
    });
  }

  render() {
    return (
      <div style={{textAlign: "center"}}>
        {this.state.msg === null ? (
          <Spin
            size="large"
            tip="Signing in..."
            style={{paddingTop: "10%"}}
          />
        ) : (
          <div style={{display: "inline"}}>
            <Result
              status="error"
              title="Login Error"
              subTitle={this.state.msg}
              extra={[
                <Button type="primary" key="details">
```

## 3. Get login URLs

Next you can show the "Sign up" and "Sign in" buttons or links to your users. The URLs can either be retrieved in the frontend or backend. See more details at: /docs/basic/core-concepts#login-urls

## 4. Get and verify access token

Here are the steps:

1. The user clicks the login URL and is redirected to Casdoor's login page, like: `https://door.casdoor.com/login/oauth/authorize?client_id=014ae4bd048734ca2dea&response_type=code&redirect_uri=https%3A%2F%2Fforum.casbin.com%2Fcallback&scope=read&state=app-casnode`
2. The user enters username & password and clicks `Sign In` (or just click the third-party login button like `Sign in with GitHub`).
3. The user is redirected back to your application with the authorization code issued by Casdoor ( like: `https://forum.casbin.com?code=xxx&state=yyy`), your application's backend needs to exchange the authorization code with the access token and verify that the access token is valid and issued by Casdoor. The functions `GetOAuthToken()` and `ParseJwtToken()` are provided by Casdoor backend SDK.

The following code shows how to get and verify the access token. For a real example of Casnode (a forum website written in Go), see: https://github.com/casbin/casnode/blob/6d4c55f5c9a3c4bd8c85f2493abad3553b9c7ac0/controllers/account.go#L51-L64

```go
// get code and state from the GET parameters of the redirected URL
code := c.Input().Get("code")
state := c.Input().Get("state")

// exchange the access token with code and state
token, err := auth.GetOAuthToken(code, state)
if err != nil {
    panic(err)
}

// verify the access token
claims, err := auth.ParseJwtToken(token.AccessToken)
if err != nil {
    panic(err)
}
```

If `ParseJwtToken()` finishes with no error, then the user has successfully logged into the application. The returned `claims` can be used to identity the user later.

## 4. Identify user with access token

> **INFO**
>
> This part is actually your application's own business logic and not part of OIDC, OAuth or Casdoor. We just provide good practices as a lot of people don't know what to do for the next step.

In Casdoor, access token is usually identical as ID token. They are the same thing. So the access token contains all information for the logged-in user.

The variable `claims` returned by `ParseJwtToken()` is defined as:

```go
type Claims struct {
    User
    AccessToken string `json:"accessToken"`
    jwt.RegisteredClaims
}
```

1. `User`: the User object, containing all information for the logged-in user, see definition at: /docs/basic/core-concepts#user
2. `AccessToken`: the access token string.
3. `jwt.RegisteredClaims`: some other values required by JWT.

At this moment, the application usually has two ways to remember the user session: `session` and `JWT`.

#### Session

The Method to set session varies greatly depending on the language and web framework. E.g., Casnode uses Beego web framework and set session by calling: `c.SetSessionUser()`.

```
token, err := auth.GetOAuthToken(code, state)
if err != nil {
    panic(err)
}

claims, err := auth.ParseJwtToken(token.AccessToken)
if err != nil {
    panic(err)
}

claims.AccessToken = token.AccessToken
c.SetSessionUser(claims) // set session
```

#### JWT

The `accessToken` returned by Casdoor is actually a JWT. So if your application uses JWT to keep user session, just use the access token directly for it:

1. Send the access token to frontend, save it in places like localStorage of the browser.
2. Let the browser send the access token to backend for every request.
3. Call `ParseJwtToken()` or your own function to verify the access token and get logged-in user information in your backend.

## 5. (Optional) Interact with the User table

> **INFO**
> This part is provided by `Casdoor Public API` and not part of the OIDC or OAuth.

Casdoor Backend SDK provides a lot of helper functions, not limited to:

- `GetUser(name string)`: get a user by username.
- `GetUsers()`: get all users.
- `AddUser()`: add a user.
- `UpdateUser()`: update a user.
- `DeleteUser()`: delete a user.
- `CheckUserPassword(auth.User)`: check user's password.

These functions are implemented by making RESTful calls against `Casdoor Public API`. If a function is not provided in Casdoor Backend SDK, you can make RESTful calls by yourself.

# How to Enable Single Sign-On

## Introduction

You have connected Casdoor and configured more than one application in an organization. You want users to sign in once to any app in nomeguythe organization, and then be able to sign in when they go to another app, without any extra clicks.

We offer this single sign-on, you just need to:

- Enable Auto signin button.
- Fill in the URL for home page.
- Add a **Silent Signin** function to the application home page.

> NOTE
>
> The basic sign in process provided by Casdoor allows users to log in to other applications in the organization by selecting the user who is currently logged in or using another account.
>
> After enable the auto signin, the selection box will not display, the logged user will log in directly.

# Configure

1. Fill the field **home**. It can be the application's home page or the login page.

2. Enable Auto signin button.

# Add Silent Signin

In fact, we implement auto login by carrying parameters on the URL. So your applications need to have a method to trigger the login after jumping to the URL. We provide casdoor-react-sdk for you to quickly implement the feature. You can see details in use-in-react.

> INFO
>
> How it works
>
> 1. In the URL to the application home page, we will carry the `silentSignin` parameter.
> 2. In your HomePage to determine whether you need to log in

> silently(automatically) by the parameter `silentSignin`. If silentSignin === 1, the function returns the SilentSignin component, it will help you initiate a login request. And since you have auto-login enabled, users will log in automatically without clicking.

# Add Popup Signin

`popup signin` will pop up a small window. After logging in to Casdoor in the child window, it will send authentication information to the main window and then close automatically. We implement it by carrying parameters on the URL.

> INFO
>
> How to use
>
> Use the method `popupSignin()` in sdk casdoor-js-sdk to quickly implement the feature. You can see a demo in casdoor-nodejs-react-example.
>
> How it works
>
> 1. In the URL to the application home page, we will carry the `popup` parameter.
> 2. When `popup=1` in login params, Casdoor will send `code` and `state` as a message to main window and finish get `token` in main window by SDK.

# Using SSO

The configuration is complete, below will show you how to use auto login.

> INFO
>
> Make sure in your application can redirect to user's profile page. The API getMyProfileUrl(account, returnUrl) is provided in our SDK for each language.

Open the profile page and go to the "Home" page (`/` URL path). You will see the application list provided by the organization. It's notable that only users in organizations other that `built-in` can see the application list in the "Home" page. All the global administrators (aka in the `built-in` organization) cannot see it.

Click on a tile in the application list, it will jump to the homepage URL of that application with GET parameter: `?silentSignin=1` and automatically log into the application if the application has integrated with Casdoor SSO (so it will recognize

the `?silentSignin=1` parameter and perform silent login in the background).

# Vue SDK

Casdoor Vue SDK is designed for Vue2 and Vue3 which is very convenient to use.

The Vue SDK is based on casdoor-js-sdk,you can also use the casdoor-js-sdk directly which will be more customizable.

This plugin is still in development. If you have any questions and suggestions, please contact us at issue

We will show you the steps below.

> if you still don't know how to use it after reading README.md, you can go to the example: casdoor-python-vue-sdk-example for more details.

> The example' front-end is built with casdoor-vue-sdk, and the back-end is built with casdoor-python-sdk, you can see the source code in the example.

## Installation

```
# NPM
npm i casdoor-vue-sdk

# Yarn
yarn add casdoor-vue-sdk
```

# Init SDK

Initialization requires 5 parameters, which are all string type:

| Name (in order) | Must | Description |
|---|---|---|
| serverUrl | Yes | your Casdoor server URL |
| clientId | Yes | the Client ID of your Casdoor application |
| appName | Yes | the name of your Casdoor application |
| organizationName | Yes | the name of the Casdoor organization connected with your Casdoor application |
| redirectPath | No | the path of the redirect URL for your Casdoor application, will be `/callback` if not provided |

install:

For Vue3:

```
// in main.js
import Casdoor from 'casdoor-vue-sdk'
const config = {
  serverUrl: "http://localhost:8000",
  clientId: "4262bea2b293539fe45e",
  organizationName: "casbin",
  appName: "app-casnode",
  redirectPath: "/callback",
```

For Vue2:

```
// in main.js
import Casdoor from 'casdoor-vue-sdk'
import VueCompositionAPI from '@vue/composition-api'
const config = {
  serverUrl: "http://localhost:8000",
  clientId: "4262bea2b293539fe45e",
  organizationName: "casbin",
  appName: "app-casnode",
  redirectPath: "/callback",
};
Vue.use(VueCompositionAPI)
Vue.use(Casdoor,config)
new Vue({
  render: h => h(App),
}).$mount('#app')
```

# example

```
// in app.vue
<script>
export default {
  name: 'App',
  methods: {
    login() {
      window.location.href = this.getSigninUrl();
    },
    signup() {
      window.location.href = this.getSignupUrl();
    }
  }
}
</script>
```

Auto Fix

If the `postinstall` hook doesn't get triggered or you have updated the Vue version, try to run the following command to resolve the redirecting.

```
npx vue-demi-fix
```

More info about Vue version switch at: vue-demi docs

# Desktop SDKs

## Electron App

An Electron app example for Casdoor

## Dotnet Desktop App

An Dotnet desktop app example for Casdoor

## Qt Desktop App

An Qt desktop app example for Casdoor

# Electron App

An Electron app example for Casdoor.

## How to run example

### Initialization

You need to initialize 6 parameters, which are all string type:

| Name | Description | Path |
|---|---|---|
| serverUrl | your Casdoor server URL | `src/App.js` |
| clientId | the Client ID of your Casdoor application | `src/App.js` |
| appName | the name of your Casdoor application | `src/App.js` |
| redirectPath | the path of the redirect URL for your Casdoor application, will be `/callback` if not provided | `src/App.js` |
| clientSecret | the Client Secret of your Casdoor application | `src/App.js` |
| casdoorServiceDomain | your Casdoor server URL | `public/electron.js` |

If you don't set these parameters, this project will use the Casdoor online demo as the default Casdoor server and use the Casnode as the default Casdoor application.

### Available commands

In the project directory, you can run:

**`npm run dev`** or **`yarn dev`**

Builds the electron app and run this app.

**`npm run make`** or **`yarn make`**

Package and distribute your application. It will create the `out` folder where your package will be located:

```
// Example for macOS out/
├── out/make/zip/darwin/x64/casdoor-electron-example-darwin-x64-1.0.0.zip
├── ...
└── out/casdoor-electron-example-darwin-x64/casdoor-electron-example.app/Contents/MacOS/casdoor-electron-example
```

### Prview

After you run this electron application, a new window will be showed on your desktop.

If you click `Login with Casdoor` button, your default browser will be opened automatically and show the login page.

After you login successfully, your electron application will be opened and your user name will be showed on your application.

You can preview the whole process by the gif image below.

# How to integrate

## Set the custom protocol

Firstly, you need to set the custom protocol called `casdoor`.

```
const protocol = "casdoor";

if (process.defaultApp) {
  if (process.argv.length >= 2) {
    app.setAsDefaultProtocolClient(protocol, process.execPath, [
      path.resolve(process.argv[1]),
    ]);
  }
} else {
  app.setAsDefaultProtocolClient(protocol);
}
```

This will help the browser to open your electron application and send the login info to the electron application.

## Open the login url by the browser

```
const serverUrl = "https://door.casdoor.com";
const appName = "app-casnode";
const redirectPath = "/callback";
const clientId = "014ae4bd048734ca2dea";
const clientSecret = "f26a4115725867b7bb7b668c81e1f8f7fae1544d";

const redirectUrl = "casdoor://localhost:3000" + redirectPath;
```

You can change the first 5 parameters.

## Listen to the open application event

After you login successfully in the browser, the browser will open your electron application. You need to listen to the open application event.

```
const gotTheLock = app.requestSingleInstanceLock();
const ProtocolRegExp = new RegExp(`^${protocol}://`);

if (!gotTheLock) {
  app.quit();
} else {
  app.on("second-instance", (event, commandLine, workingDirectory) => {
    if (mainWindow) {
      if (mainWindow.isMinimized()) mainWindow.restore();
      mainWindow.focus();
      commandLine.forEach((str) => {
        if (ProtocolRegExp.test(str)) {
          const params = url.parse(str, true).query;
          if (params && params.code) {
            store.set("casdoor_code", params.code);
            mainWindow.webContents.send("receiveCode", params.code);
          }
        }
      });
    }
  });
  app.whenReady().then(createWindow);

  app.on("open-url", (event, openUrl) => {
    const isProtocol = ProtocolRegExp.test(openUrl);
    if (isProtocol) {
      const params = url.parse(openUrl, true).query;
      if (params && params.code) {
        store.set("casdoor_code", params.code);
        mainWindow.webContents.send("receiveCode", params.code);
      }
    }
  });
}
```

You can get the code from the browser, which is `casdoor_code` or `params.code`.

## Parse the code and get the user info

```
async function getUserInfo(clientId, clientSecret, code) {
  const { data } = await axios({
    method: "post",
    url: authCodeUrl,
    headers: {
      "content-type": "application/json",
    },
    data: JSON.stringify({
      grant_type: "authorization_code",
      client_id: clientId,
      client_secret: clientSecret,
      code: code,
    }),
  });
  const resp = await axios({
    method: "get",
    url: `${getUserInfoUrl}?accessToken=${data.access_token}`,
  });
  return resp.data;
```

Finally, you can parse the code and get the user info just by following the OAuth docs page.

# Dotnet Desktop App

An Dotnet desktop app example for Casdoor.

## How to run example

## Prerequisites

dotnet6 sdk

webview2 runtime (It's already preinstalled in your windows generally)

## Initialization

The initialization requires 5 parameters, which are all string type:

| Name | Description | File |
| --- | --- | --- |
| Domain | Your Casdoor server host/domain | `CasdoorVariables.cs` |
| ClientId | The Client ID of your Casdoor application | `CasdoorVariables.cs` |
| AppName | The name of your Casdoor application | `CasdoorVariables.cs` |

| Name | Description | File |
| --- | --- | --- |
| CallbackUrl | The path of the callback URL for your Casdoor application, will be `casdoor://callback` if not provided | `CasdoorVariables.cs` |
| ClientSecret | The Client Secret of your Casdoor application | `CasdoorVariables.cs` |

If you don't set these parameters, this project will use the Casdoor online demo as the default Casdoor server and use the Casnode as the default Casdoor application.

# Running

## Visual Studio

1. Open casdoor-dotnet-desktop-example.sln
2. Press Ctrl + F5 to start

## Command line

1. cd src/DesktopApp
2. dotnet run

# Preview

After you run this dotnet desktop application, a new window will be showed on

your desktop.

If you click `Casdoor Login` button, a login window will be showed on your

Login
casbin
密码
WebAuthn
admin
下次自动登录
忘记密码?
登 录
没有账号? 立即注册
Feedback
Made with by Casdoor

desktop.

After you login successfully, a user profile window will be showed on your desktop. It displays your user name.

You can preview the whole process by the gif image below.

# How to integrate

# Open the login window

```
var login = new Login();
// Trigger when login succeeded, you will receive auth code in
event handler
login.CodeReceived += Login_CodeReceived;
login.ShowDialog();
```

# Use auth code to get the user info

```
public async Task<string?> RequestToken(string clientId, string
clientSecret, string code)
{
    var body = new
    {
        grant_type = "authorization_code",
        client_id = clientId,
        client_secret = clientSecret,
        code
    };

    var req = new RestRequest(_requestTokenUrl).AddJsonBody(body);
    var token = await _client.PostAsync<TokenDto>(req);

    return token?.AccessToken;
}

public async Task<UserDto?> GetUserInfo(string token)
{
    var req = new
RestRequest(_getUserInfoUrl).AddQueryParameter("accessToken",
```

# Qt Desktop App

A Qt desktop app example for Casdoor.

## How to run example

### Prerequisites

Qt6 sdk

OpenSSL toolkit

### Initialization

You need to initialize 7 parameters, which are all string type:

| Name | Description | File |
| --- | --- | --- |
| endpoint | Your Casdoor server host/domain | `mainwindow.h` |
| client_id | The Client ID of your Casdoor application | `mainwindow.h` |
| client_secret | The Client Secret of your Casdoor application | `mainwindow.h` |
| certificate | The public key for the Casdoor application's cert | `mainwindow.h` |

| Name | Description | File |
| --- | --- | --- |
| org_name | The name of your Casdoor organization | `mainwindow.h` |
| app_name | The name of your Casdoor application | `mainwindow.h` |
| redirect_url | The path of the callback URL for your Casdoor application, will be `http://localhost:8080/callback` if not provided | `mainwindow.h` |

If you don't set the parameter `endpoint`, this project will use the http://localhost:8000 as the defult Casdoor server.

## Running

### Qt Creator

1. Open casdoor-cpp-qt-example.pro
2. Set the `INCLUDEPATH` of OpenSSL in casdoor-cpp-qt-example.pro
3. Press Ctrl + R to start

## Preview

After you run this Qt desktop application, a new window will be showed on your desktop.

If you click `Sign In` botton, a login window will be showed on your desktop.

After you login successfully, a user profile window will be showed on your desktop, it dispaly your user information.

You can preview the whole process by the gif image below.

# How to integrate

## Open the login window

```
// Load and display the login page of Casdoor
m_webview->page()->load(*m_signin_url);
m_webview->show();
```

## Listen to the open application event

```
// Initialize the TcpServer object and listen on port 8080
m_tcpserver = new QTcpServer(this);
if(!m_tcpserver->listen(QHostAddress::LocalHost, 8080)) {
    qDebug() << m_tcpserver->errorString();
    close();
}
connect(m_tcpserver, SIGNAL(newConnection()), this,
SLOT(on_tcp_connected()));
```

## Use auth code to get the user info

```
// Get token and parse it with the JWT library
std::string token = m_casdoor->GetOAuthToken(code.toStdString());
auto decoded = m_casdoor->ParseJwtToken(token);
```

# Casdoor Plugin

Casdoor also provides plugins or middlewares for some very popular platforms, like Java's SpringBoot, PHP's WordPress, Python's Odoo etc.

| Casdoor plugin | Language | Source code |
|---|---|---|
| Spring Boot plugin | Java | https://github.com/casdoor/casdoor-spring-boot-starter |
| Spring Boot example | Java | https://github.com/casdoor/casdoor-spring-boot-example |
| WordPress plugin | PHP | https://github.com/casdoor/wordpress-casdoor-plugin |
| Odoo plugin | Python | https://github.com/casdoor/odoo-casdoor-oauth |
| Django plugin | Python | https://github.com/casdoor/django-casdoor-auth |

For a full list of the official Casdoor plugins, please see: Casdoor repositories.

# OAuth 2.0

## Introduction

Casdoor supports AccessToken to authenticate clients. In this section, we will show you how to get AccessToken, how to verify AccessToken and how to use AccessToken.

## How to get AccessToken

You have two ways to get the AccessToken: you can use the Casdoor SDKs, for details please refer to the SDK documentation, here we will mainly show you how to use the API to get the AccessToken.

Casdoor supports four OAuth grant types: Authorization Code Grant, Implicit Grant, Resource Owner Password Credentials Grant, and Client Credentials Grant.

For security reasons, the Casdoor app has the authorization code mode turned on by default. If you need to use other modes, please go to the appropriate app to set it.

### Authorization Code Grant

First redirect your users to:

```
https://<CASDOOR_HOST>/login/oauth/authorize?
client_id=CLIENT_ID&
redirect_uri=REDIRECT_URI&
response_type=code&
scope=openid&
state=STATE
```

Available scopes

| Name | Description |
| --- | --- |
| openid (no scope) | sub (user's id), iss (issuer) and aud (audience) |
| profile | user profile info, include name, displayName, avatar |
| email | user's email address |
| address | user's address |
| phone | user's phone number |

> INFO
>
> Your OAuth Application can request the scopes in the initial redirection. You can specify multiple scopes by separating them with a space using %20:
>
> ```
> https://<CASDOOR_HOST>/login/oauth/authorize?
> client_id=...&
> scope=openid%20email
> ```
>
> For more details, please see OIDC standard

After your user has authenticated with casdoor, casdoor will redirect him to:

```
https://REDIRECT_URI?code=CODE&state=STATE
```

Now that you have obtained the authorization code, make a POST request to:

```
https://<CASDOOR_HOST>/api/login/oauth/access_token
```

in your backend application:

```
{
    "grant_type": "authorization_code",
    "client_id": ClientId,
    "client_secret": ClientSecret,
    "code": Code,
}
```

You will get the following response:

```
{
    "access_token": "eyJhb...",
    "id_token": "eyJhb...",
    "refresh_token": "eyJhb...",
    "token_type": "Bearer",
    "expires_in": 10080,
    "scope": "openid"
}
```

> NOTE
>
> Casdoor also supports PKCE feature, when getting the authorization code, you can add two parameters to enable PKCE:
>
> ```
> &code_challenge_method=S256&code_challenge=YOUR_CHANLLENGE
> ```
>
> When getting the token you need to pass `code_verifier` parameter to verify PKCE. It is worth mentioning that with PKCE enabled, Client_Secret is not required, but if you pass it, it must be the correct value.

## Implicit Grant

Maybe your application doesn't have a backend, and you need to use Implicit Grant. First you need to make sure you have Implicit Grant enabled, then redirect your users to:

```
https://<CASDOOR_HOST>/login/oauth/
authorize?client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=token&scope=openid&state=STATE
```

After your user has authenticated with casdoor, casdoor will redirect him to:

```
https://REDIRECT_URI/#access_token=ACCESS_TOKEN
```

Casdoor also supports id_token as `response_type`, which is a feature of OpenID.

## Resource Owner Password Credentials Grant

If your application doesn't have a frontend that redirects users to Casdoor, then you may need this.

First you need to make sure you have Password Credentials Grant enabled, and send a POST request to:

```
https://<CASDOOR_HOST>/api/login/oauth/access_token
```

```
{
    "grant_type": "password",
    "client_id": ClientId,
```

You will get the following response:

```
{
    "access_token": "eyJhb...",
    "id_token": "eyJhb...",
    "refresh_token": "eyJhb...",
    "token_type": "Bearer",
    "expires_in": 10080,
    "scope": "openid"
}
```

## Client Credentials Grant

You can also use Client Credentials Grant when your application does not have a frontend.

First you need to make sure you have Client Credentials Grant enabled, and send a POST request to `https://<CASDOOR_HOST>/api/login/oauth/access_token`:

```
{
    "grant_type": "client_credentials",
    "client_id": ClientId,
    "client_secret": ClientSecret,
}
```

You will get the following response:

```
{
    "access_token": "eyJhb...",
    "id_token": "eyJhb...",
    "refresh_token": "eyJhb...",
    "token_type": "Bearer",
    "expires_in": 10080,
    "scope": "openid"
}
```

It is important to note that the AccessToken obtained in this way differs from the first three in that it corresponds to the application rather than to the user.

## RefreshToken

Maybe you want to update your accessToken, then you can use the `refreshToken` you got above.

First you need to set the expiration time of refreshToken in the application (default is 0 hours), and send a POST request to `https://<CASDOOR_HOST>/api/login/oauth/refresh_token`

```
{
    "grant_type": "refresh_token",
    "refresh_token": REFRESH_TOKEN,
```

You will get the response like:

```
{
    "access_token": "eyJhb...",
    "id_token": "eyJhb...",
    "refresh_token": "eyJhb...",
    "token_type": "Bearer",
    "expires_in": 10080,
    "scope": "openid"
}
```

# How to verify AccessToken

Casdoor currently has support for token introspection endpoint. Currently the endpoint is protected by Basic Authoritarian(ClientId:ClientSecret):

```
POST /api/login/oauth/introspect HTTP/1.1
Host: CASDOOR_HOST
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ=

token=ACCESS_TOKEN&token_type_hint=access_token
```

You will get the following response like:

```
{
    "active": true,
    "client_id": "c58c...",
    "username": "admin",
    "token_type": "Bearer",
    "exp": 1647138242,
    "iat": 1646533442,
    "nbf": 1646533442,
    "sub": "7a6b4a8a-b731-48da-bc44-36ae27338817",
    "aud": [
        "c58c..."
    ],
    "iss": "http://localhost:8000"
}
```

# How to use `AccessToken`

You can use AccessToken to access Casdoor APIs that require authentication.

For example, two different ways to request `/api/userinfo`.

Type 1. Query parameter

`https://<CASDOOR_HOST>/api/userinfo?accessToken=<your_access_token>`

Type 2. HTTP Bearer token

`https://<CASDOOR_HOST>/api/userinfo` with the header: "Authorization: Bearer <your_access_token>"

Casdoor will parse the access_token, returning corresponding user information according to the `scope`. You will get the same response like:

```
{
    "sub": "7a6b4a8a-b731-48da-bc44-36ae27338817",
    "iss": "http://localhost:8000",
    "aud": "c58c..."
}
```

If you expect more user's information, adding `scope` when getting AccessToken in step Authorization Code Grant.

# Differences between `userinfo` and `get-account` APIs

- `/api/userinfo`: returns user information as part of OIDC protocol. Less information is returned, including only the basic information in OIDC standards. Please see available scopes that Casdoor supports.
- `/api/get-account`: gets the user object for the currently logged-in account. It is a Casdoor-only API to obtain all information of the user in Casdoor.

# CAS

## Using Casdoor as CAS server

Casdoor now can be used as CAS server. Up to now the casdoor has supported the feature of CAS3.0 .

### Overview

The prefix of CAS endpoint in Casdoor is `<Endpoint of casdoor>/cas/<organization name>/<application name>`,which means:

Suppose the endpoint of Casdoor is `https://door.casdoor.com`, which contains an application called `cas-java-app` which belongs to an organization called `casbin`, and if we are trying to let user login in via CAS, then

- `/login` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/login`
- `/logout` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/logout`
- `/serviceValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/serviceValidate`
- `/proxyValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/proxyValidate`
- `/proxy` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/proxy`
- `/validate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/validate`
- `/p3/serviceValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/p3/serviceValidate`
- `/p3/proxyValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/p3/proxyValidate`
- `/samlValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/samlValidate`

See https://apereo.github.io/cas/6.6.x/protocol/CAS-Protocol-Specification.html for more information about CAS and its different versions, as well as parameters for these endpoints.

### An example

Here is an offical example https://github.com/apereo/cas-sample-java-webapp, which contains an example web app utilizing the offical CAS java client https://github.com/apereo/java-cas-client. By going through this example, we will illustrate how to connect to Casdoor via CAS.

> NOTE
>
> Note: Currently Casdoor only support all three versions of CAS: CAS 1.0 & 2.0 & CAS 3.0 .

The cas configuration is located in `src/main/webapp/WEB-INF/web.yml`.

By default, this app uses CAS 3.0, which is specified by the following configurations.

```
<filter-name>CAS Validation Filter</filter-name>
<filter-
class>org.jasig.cas.client.validation.Cas30ProxyReceivingTicketValidationFilter</filter-
class>
```

Suppose you want to protect this web app via CAS 2.0, you are supposed to change CAS Validation Filter to the following content.

```
<filter-name>CAS Validation Filter</filter-name>
<filter-
class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter</filter-
class>
```

If you want to use CAS 1.0, use

```
<filter-name>CAS Validation Filter</filter-name>
<filter-class>org.jasig.cas.client.validation.Cas10TicketValidationFilter</filter-class>
```

For all the appearances of parameter 'casServerUrlPrefix', change them to

```
<param-name>casServerUrlPrefix</param-name>
<param-value>http://door.casdoor.com/cas/casbin/cas-java-app</param-value>
```

For all the appearances of parameter 'casServerLoginUrl' change them to

```
<param-name>casServerLoginUrl</param-name>
<param-value>http://door.casdoor.com/cas/casbin/cas-java-app/login</param-value>
```

If you need to customize more configurations, see https://github.com/apereo/java-cas-client for detailed information.

> **NOTE**
>
> Actually, we have already have this demo app run on https://cas-java-app.casdoor.com. You can visit here to experience using Casdoor via CAS.

# SAML

## Overview

Using Casdoor as SAML IdP

## Keycloak

Using Casdoor as SAML IdP

## Appgate (POST)

How to use Casdoor as SAML IdP for Appgate

# Overview

Casdoor now can be used as SAML IdP. Up to now the Casdoor has supported the main feature of SAML 2.0 .

## Configuration in SP

Generally, SP needs the `Single Sign-On`, `Issuer` and `Public Certificate` three required fields. Most of the SP can get these fields by uploading the XML Metadata file or the XML Metadata URL to autocomplete.

The metadata of SAML endpoint in Casdoor is `<Endpoint of casdoor>/api/saml/metadata?application=admin/<application name>`. Suppose the endpoint of Casdoor is `https://door.casdoor.com`, which contains an application called `app-built-in`. The XML Metadata endpoint will be:

```
https://door.casdoor.com/api/saml/metadata?application=admin/app-built-in
```

And you can also find the metadata in the application edit page. Click the button to copy the URL and paste it in browser to download the XML Metadata.

## Configuration in Casdoor IdP

Casdoor supports both GET and POST `SAMLResponse`. Casdoor need to know what types of request the SP supports, when Casdoor sends the `SAMLResponse` to SP. You need to configure the application in casdoor based on the `SAMLResponse` type supported by your SP.

> INFO
>
> If you fill the `Reply URL`, Casdoor will send the `SAMLResponse` by **POST** Request. If the Reply URL is empty, Casdoor will use **GET** request. You might wonder how casdoor knows the `Reply URL` of the SP, if the `Reply URL` is empty. Actually, Casdoor can get the URL called `AssertionConsumerServiceURL` by parsing the `SAMLRequest`. And send the request with `SAMLResponse` to `AssertionConsumerServiceURL`.
>
> The `Reply URL` will overwrites the `AssertionConsumerServiceURL` in `SAMLRequest`.

- **Reply URL** Type in the URL of the ACS verifying the SAML response.

- **Redirect URL** Type in a unique name. This may be called `Audience` or `Entity`

`ID` in your SP. Make sure you fill the same `Redirect URL` here as in your SP.

## User profile

After logged in successfully, the user profile in the `SAMLResponse` Casdoor returned has three fields. The attributes in the xml and the attributes of the user in casdoor are mapped as follows:

| XML Attribute Name | User field |
|---|---|
| Email | email |
| DisplayName | displayName |
| Name | name |

See https://en.wikipedia.org/wiki/SAML_2.0 for more information about SAML and its different versions.

## An example

The gosaml2 is a SAML 2.0 implementation for Service Providers based on etree

and goxmldsig, a pure Go implementation of XML digital signatures. And we use this library to test the SAML 2.0 in Casdoor as below.

Suppose you can access Casdoor through `http://localhost:7001/`, and your Casdoor contains an application called `app-built-in` which belongs to an organization called `built-in`. The URLs, `http://localhost:6900/acs/example` and `http://localhost:6900/saml/acs/example`, should be added to the Redirect URLs in `app-built-in`.

```
import (
    "crypto/x509"
    "fmt"
    "net/http"

    "io/ioutil"

    "encoding/base64"
    "encoding/xml"

    saml2 "github.com/russellhaering/gosaml2"
    "github.com/russellhaering/gosaml2/types"
    dsig "github.com/russellhaering/goxmldsig"
)

func main() {
    res, err := http.Get("http://localhost:7001/api/saml/
metadata?application=admin/app-built-in")
    if err != nil {
        panic(err)
    }

    rawMetadata, err := ioutil.ReadAll(res.Body)
    if err != nil {
        panic(err)
    }
```

Run the above codes and the console will display the following message.

```
Visit this URL To Authenticate:
http://localhost:7001/login/saml/authorize/admin/app-built-
in?SAMLRequest=lFVbk6K8Fv0rFvNo2QR...
Supply:
  SP ACS URL      : http://localhost:6900/v1/_saml_callback
```

Click the URL to authenticate, the login page of Casdoor will display.

You will get the response messages as below after authenticating.

# Keycloak

## Casdoor as a SAML IdP in keycloak

This guide will show you how to configure Casdoor and Keycloak to add Casdoor as a SAML IdP in Keycloak.

### Add SAML IdP in Keycloak

Open Keycloak admin page, click **Identity Providers** and select **SAML v2.0** from the list of providers.

> **INFO**
>
> You can visit Keycloak SAML Identity Providers documentation to get more detailed information.

Enter the **Alias** and the **Import from URL** in Keycloak IdP edit page. The content of **Import from URL** can be found in the Casdoor application edit page. Click **Import** and the SAML config will be filled automatically.

You should remember the **Service Provider Entity ID** and then save the configuration.

### Configure SAML application in Casdoor

In the application edit page, add a redirect URL which the content of it is **Service Provider Entity ID** in Keycloak. And you should enable SAML compress for Keycloak.

## Login using Casdoor SAML

Open the Keycloak login page and you can find the additional button that allows you to login to Keycloak using the Casdoor SAML provider.

Click on the button and you will be redirected to the Casdoor SAML provider for the authentication. After the successful authentication, you will be redirected back to Keycloak. Then you need to assign users to the application.

We also provide a demo video to demonstrate the entire process, which we hope will be helpful to you.

# Appgate (POST)

## Casdoor as a SAML IdP in Appgate

Appgate accepts the `SAMLResponse` sent by POST Request. If you use other SP that also supports POST request, you can refer to this document.

### Casdoor configuration

Go to your Casdoor and add a new application.

Enter basic SAML configuration in the application:

- Redirect URLs – Type in a unique name. This may be called `Audience` or `Entity ID` in your SP. See the table below.

- Reply URL – type in the URL of the ACS verifying the SAML response, refer to the table below

| Administrator Authentication | User Authentication |
| --- | --- |
| Redirect URL = “AppGate” | Redirect URL = “AppGate Client” |
| SAML Reply URL = https://mycontroller.mycompany.com/admin/saml | SAML Reply URL = https://redirectserver.mycompany.com/saml |

## Download the XML metadata file

You can copy the URL of metadata and download the file from your browser.

# Add SAML IdP in Appgate

In your AppGate SDP console:

- Select System > Identity Providers
- Create a new Identity Provider
- Choose the type SAML
- Start configuring your identity provider following the details in the tables below

| | Administrator Authentication |
|---|---|
| Name | Enter a unique name eg: "Casdoor SAML Admin" |
| Single Sign-on URL | See below |
| Issuer | See below |
| Audience | Type in the **Redirect URL** from the Casdoor application |
| Public Certificate | See below |

- Upload the XML Metadata file to autocomplete **Single Sign-On**, **Issuer** and **Public Certificate fields**
- Click **Choose a file** and select the metadata file that you created previously - this will autocomplete the relevant fields

## Map Attributes

Mapping the **Name** to **username**, your completed form should look something like this:

## Test Integration

On your AppGate SDP Controller console:

- Log out of the admin UI
- Log in using the following information:
- Identity Provider – choose your Azure IdP from the drop down list
- Click **Sign in with browser** to connect to your authenticator
- You may see the following message: “You don’t have any administration rights” – this confirms that the test user credentials have been successfully authenticated by your Identity Provider.

## Access Policy

Your need to modify the access policy that the administrator can log in the Appgate by the SAML idp. Enter Builtin Administrator Policy:

Your completed form should look something like this:

Home > Access – Policies > Edit

# Editing Policy - Admin

- (●) Enabled
- ( ) Disabled

**Assignment** - Active when **custom logic is met**

Add New

**Custom Logic** (1 OR 3) AND 2

| | |
|---|---|
| 1 | Identity Provider is **local** |
| 2 | **user.username** is **admin** |
| 3 | Identity Provider is **Casdoor SAML Admin** |

# WebAuthn

## Overview

We are delighted to inform the Casdoor's customers that Casdoor now supports logging in with WebAuthn, which means, you may be able to log in with your biological identifications like fingerprints or facial recognition even U-disks, provided that your device support these cool authorization method and WebAuthn.

### What is WebAuthn?

The Web Authentication API (also known as WebAuthn: https://webauthn.io/) is a specification written by the W3C and FIDO, with the participation of Google, Mozilla, Microsoft, Yubico, and others. The API allows servers to register and authenticate users using public key cryptography instead of a password. **It allows servers to integrate with the strong authenticators now built into devices, like Windows Hello or Apple's Touch ID.**

To be concise, WebAuthn asks users to generate a public key - private key pair, and hand over the public key to the website. When a user wants to log in to a website, the web generates a random number and asks the user to encrypt it with its private key and send back the result. After receiving the result, the website will try to use the public key to decrypt, and if the decrypted number is the same as the random number generated before, the user will be regarded as a legal user and he will be allowed to log in. We call the public key combined with necessary information (like username or information about user's authorizer ) the Webauthn Credential, which is exactly what is stored by the website.

The public key - private key pair is exclusively uniquely distinguished three information: (user's username, user's authorizer, and the website's url). This means, if the (user's username, user's authorizer, and the website's url) is all the same, the key pair should be identical, and vice versa.

For more detailed information about the WebAuthn Technology, you can visit https://webauthn.guide/.

## How to use WebAuthn in casdoor?

In the login page, you must have already seen the choice of using WebAuthn to login in. But considering that you haven't got a Webauthn credential (webauth password, if this inaccurate explanation can make you understand better) yet, so in this tutorial, we are going to show you how to create and manage a credential first and then, how to log in with the credential.

#### Step0: modify the configurations and turn on the WebAuthn authentication

In conf/app.conf you can see

```
origin = "http://localhost:8000"
```

Please ensure this configuration is EXACTLY the url of your website

#### Only https is supported for WebAuthn unless you are using localhost

Then log in as the administrator and go to the edit page of your application. Turn the switch on "Enable WebAuthn signin". By default, this feature is not enabled.

#### Step 1: go to "my account" page

Step 1: go to account page. On this page, you shall see the "Add WebAuthn Credential" Button and a list manifesting all the Webauthn credentials you have

previously registered.

Press the button and then follow the instructions of your device to register a new credential into casdoor.

You can remove any credentials via the "delete" button in the list.

### Step 2: log in via WebAuthn

Before this step starts, make sure you have logged out the casdoor.

Go to the log in page, choose the wenauthn login method, enter your username and press the login button, and follow the instructions of your device.

(For example, if you use fingerprint and Window Hello, you are supposed to see something like this)

Then you will see that you have already logged in.

# Developer Guide

## Frontend

Casdoor frontend development guide

## Generating Swagger Files

Generating Swagger Files

# Frontend

The source code for Casdoor's frontend is inside the `/web` folder: https://github.com/casdoor/casdoor/tree/master/web

It is a Create-React-App (CRA) project, which has a classic CRA folder structure as follows:

| File/Directory | Description |
|---|---|
| public | the HTML root file for React |
| src | source code |
| craco.config.js | the Craco config file, can change the theme color (blue by default) here |
| crowdin.yml | Crowdin i18n config file |
| package.json | NPM/Yarn dependency file |
| yarn.lock | Yarn lock file |

Inside `/src`, there are several important files or folders as follows:

| File/Directory | Description |
|---|---|
| account | the "My profile" page for logged-in users |

| File/Directory | Description |
| --- | --- |
| auth | all code related to authentication, like OAuth, SAML, sign up page, sign in page, forget password page, etc. |
| backend | the SDK for calling Go backend API, contains all the `fetch()` calls |
| basic | the homepage (dashboard page) for Casdoor, it contains several card widgets |
| common | shared UI widgets |
| locales | i18n translation files in JSON, synced with our Crowdin project: https://crowdin.com/project/casdoor-site |
| App.js | the entrance JS file, containing all routes |
| Setting.js | the utility functions used by other code |
| OrganizationListPage.js | the page for the organization list, similar to all other XXXListPage.js |
| OrganizationEditPage.js | the page for editing one organization, similar to all other XXXEditPage.js |

# Generating Swagger Files

## Overview

As we know, beego framework provides support for generateing swagger file in order to clearify the api via the command line tool called "bee". Casdoor is built based on beego too. However, we found that the swagger files generated by bee failed to categorize the apis with "@Tag" label, so we modified the original bee to implement the function.

## How to write the comment

Most rules are exactly identical to the original bee comment formats, and the only discrepancy is that the api shall be divided into different groups according to the "@Tag" label, therefore developers are obliged to ensure that this tag is correctly added. Here is an example:

```
// @Title Login
// @Tag Login API
// @Description login
// @Param   oAuthParams     query    string  true        "oAuth
parameters"
// @Param   body    body   RequestForm  true        "Login
information"
// @Success 200 {object} controllers.api_controller.Response The
Response object
// @router /login [post]
func (c *ApiController) Login() {
```

api with same "@Tag" labels will be put into the same group.

# How to generaete the swagger file

0. write comments for api in correct format
1. fetch this repository https://github.com/casbin/bee
2. build the modified bee, for example, in the root directory of casbin/bee, run

```
go build -o mybee .
```

3. copy mybee to the base directory of casdoor
4. in that directory, run

```
mybee generate docs
```

Then you will find the new swagger files are generated.

# Organizations

## Overview

Casdoor basic unit — organization

## Account Customization

Customizing users' account items

## Customize theme

Customize themes for organizations and applications under the organization

## Manage Multi-Factor authentication items

Config Muti-Factor authentication items in organization

# Overview

Organization is the basic unit of Casdoor, which manages users and applications. If a user signed in to an organization, then he can access all applications belonging to the organization without signing in again.

In the config of applications and providers, choosing an organization is important, it determines whether a user can access the application using specific providers.

We can also set up LDAP in Casdoor. For more details, please see LDAP.

Casdoor provides multiple password storage algorithms that can be selected in the organization edit page.

| Name | Algorithm | Description | Scenario |
|---|---|---|---|
| plain | - | The password will be stored in cleartext. (default) | - |
| salt | SHA256 | SHA-256 is a patented cryptographic hash function that outputs a value that is 256 bits long. | - |
| md5-salt | MD5 | The MD5 message-digest algorithm is a cryptographically broken but still widely used hash function producing a 128-bit hash value. | Discuz! |

| Name | Algorithm | Description | Scenario |
| --- | --- | --- | --- |
| bcrypt | bcrypt | bcrypt is a password-hashing function and is used to hash and salt passwords securely. | Spring Boot, WordPress |
| pbkdf2-salt | SHA256 and PBKDF2 | PBKDF2 is a simple cryptographic key derivation function, which is resistant to dictionary attacks and rainbow table attacks. It's originally implemented in Casdoor for Keycloak syncer. Select this option if you import users by Keycloak syncer. | Keycloak |

TIP

Besides logging into Casdoor via an application (which redirects to Casdoor for SSO), a Casdoor user can also choose to directly log into Casdoor via the organization's login page: `/login/<organization_name>`, e.g., https://door.casdoor.com/login/casbin in the demo site.

# Account Customization

## Introduction

In an organization, you can customize users' **account items**. This includes whether each item is **visible**. If visible, its **view rule** and **modify rule**.

When you customize account items in an organization, this configuration takes effect on the home page of all members of that organization.

## How to customize?

Account item has four attributes:

| Column Name | Selectable Value | Description |
|---|---|---|
| Name | - | Account item name. |
| Visible | `True` / `False` | Select whether this account item is visible on the user home page. |
| ViewRule | `Rule Items` | Select a rule to use with view the account item. |
| ModifyRule | `Rule Items` | Select a rule to use with modify the account item. |

Enter the Organization Edit page, you will find the following:

Casdoor provides very simple operations to configure:

- Set the item to be visible or invisible

- Set viewing and modifying rules

There are 3 rules:

- `Public`: Everyone has permission
- `Self`: The users has their own permission
- `Admin`: The administrator has permission

# Account table

The following are all the fields in account item. For a description, you can see user.

- `Organization`
- `ID`
- `Name`
- `Display name`
- `Avatar`
- `User type`
- `Password`
- `Email`
- `Phone`
- `Country/Region`

- `Location`
- `Affiliation`
- `Title`
- `Homepage`
- `Bio`
- `Tag`
- `Signup application`
- `3rd-party logins`
- `Properties`
- `Is admin`
- `Is global admin`
- `Is forbidden`
- `Is deleted`

# Customize theme

Casdoor allows you to customize theme to satisfy UI diversity from business or brand requirements, including primary color, border radius.

In Casdoor, the scope of theme includes global, organization, and application.

1. Global scope: this is the default theme of Casdoor and is applied to any organization that chooses to follow the global theme. It can only be modified in Casdoor source code, there is no way to modify it in web UI.

2. Organization scope: the theme for an organization can be customized in the organization edit page. The theme takes effect in all the Casdoor after-login pages for the users in the organization and the entry pages (signup, signin, forget password, etc.) of the applications that follow the organization theme.

3. Application scope: the theme for an application can be customized in the application edit page. The theme takes effect in the the entry pages (signup, signin, forget password, etc.) of the application.

## Customize organization theme

We provide a demo to demonstrate how to config theme in organization:

> **INFO**
>
> If your account organization is same as the organization you are editing, after you save the configuration, it will take effect immediately as the video above show. But if they are different, you need to log in the organization to see the effect.

# Customize application theme

Applications customize theme use the same theme editor as the organization. But even more conveniently, you can preview the theme in the preview panel.

Preview :

Copy signup page URL

Copy signin page URL

Background URL

# Manage Multi-Factor authentication items

## Add Muti-Factor authentication item in organization

In organization, admin can add the Multi-factor authentication item in account items so that user can config the Multi-factor authentication in their owner profile page.

# Manage Multi-Factor authentication items

You can manage Multi-Factor authentication to determine which Multi-Factor authentication method are available to users.

There are two rules for managing Multi-Factor authentication items:

- optional: Users can choose whether to enable this type Multi-Factor authentication.
- required: Users must enable this type Multi-Factor authentication .

Here is a demo video to demonstrate the entire process, which we hope will be helpful to you.

feat: Added new locales pt-br b
Built-in Organization
localhost:8000/organizations/forum
Roles
Public
Immutable
Permissions
Public
Immutable
3rd-party logins
Self
Self
Properties
Is admin
Admin
Admin
Is global admin
Admin
Admin
Is forbidden
Admin
Admin
Is deleted
Admin
Admin
WebAuthn credentials
Self
Self
Managed accounts
Self
Self
MFA items :
MFA items
Add
Name
Rule
Action
No data
Theme :
Follow global theme
Customize theme
Theme

# Applications

## Overview

Casdoor application overview

## Terminology reference

Terminology reference

## Application Config

Configure your applications authentication

## Signup Items Table

configure the signup items table to create a custom registration page

## 📄 Login UI Customization

Customize the login page UI for your application

# Overview

Every application in Casdoor is called an `application`, and they are not related and do not affect each other, which means you can deploy or stop any application separately, as long as you like.

If you want to use Casdoor to provide login service for your web Web APPs, you can add them as Casdoor applications.

Users can access all applications in their organizations without login twice.

The application configuration is very flexible and simple. You can set whether to allow password login or third-party login, configure the third-party applications you want users to log in, and you can even customize the signup items of the application, etc.

In this chapter you will learn how to start an application of your own, everything from scratch.

Let's explore together!

# Terminology reference

- `Name` The name of the created app
- `CreatedTime` The time when the application is created
- `DisplayName` The name which the application display to public
- `Logo` Application logos will display on the login and sign up page
- `HomepageUrl` The url of the application homepage
- `Description` Describe the application
- `Organization` The organization that the APP belongs to
- `EnablePassword` If users can login via password
- `EnableSignUp` If users can sign up. If not, accounts of the application
- `SignupItems` fields that need to be filled in when users register
- `Providers` Provide all kinds of services for the applications (such as OAuth, Email, SMS service)
- `ClientId` OAuth client id
- `ClientSecret` OAuth client secret
- `RedirectUris` Casdoor will navigate to one of the uris if user logged in successfully
- `TokenFormat`: The format of the generated token. It can be either `JWT` (containing all `User` fields) or `JWT-Empty` containing all non-empty values
- `ExpireInHours` Login will expire after hours
- `SigninUrl`
- `SignupUrl` If you provide a sign up service independently out of Casdoor, please fill the url here
- `ForgetUrl` Same as `SignupUrl`
- `AffiliationUrl`

# Application Config

After you deploy your casdoor on your server, and setup your organization, you can deploy your applications now!

Let's see how to config your applications authentication using Casdoor!

> NOTE
>
> Here, for example, I want to setup my Forum using Casnode

I create my application and fill some necessary configures.

Select organization I created to make users in this organization can use this application.

While this organization is named `my_organization`, so I choose it in drop-down menu.

Then I want my users can use Casdoor to complete authentication when they are signing up, so I fill the redirect url here as https://my-site-url.com/callback

> ⚠ CAUTION
>
> So here, we need to remember the `callback URL` in provider application is **Casdoor's callback url**, and the `Redirect URL` in Casdoor is **your website callback url**
>
> **Further understanding**
>
> If I want the authentication progress to work, the detailed progress should

> be like this:
>
> Users send a request to Casdoor, Casdoor use the `Client ID` and `Client Secret` to get authentication from GitHub, Google or other providers.
>
> If the authentication success, GitHub callback to Casdoor to tell Casdoor authentication success, so the GitHub authorization callback URL should be my Casdoor callback URL which is http://your-casdoor-url.com/callback, then Casdoor tells the application authentication success which means the Casdoor callback URL should be my application callback URL, that is http://your-site-url.com/callback.

Then you can add which third party apps can sign up by adding providers and setting its properties.

You need to enable JavaScript to run this app.

TIP

Note that if you don't want users to access your app using a **username/password**, you can switch off the `Password On` button, so users can only access app using third party services:

Token expire :
168
Hours
Password ON :
Enable signup :

# Signup Items Table

On the application configuration page, we can configure the signup items table to create a customized registration page. And we can add or delete any signup item on this signup items table.

For a detailed explanation of each signup item, please see the table below.

| Column Name | Selectable Value | Description |
|---|---|---|
| Name | - | Signup item name. |
| visible | `True` / `False` | Select whether this signup item is visible on the registration page. |
| required | `True` / `False` | Select whether this signup item is mandatorily required. |
| prompted | `True` / `False` | Select whether to give a prompt when user forget to fill in this signup item. |

| Column Name | Selectable Value | Description |
| --- | --- | --- |
| rule | `Rule Items` | Select a rule to use with this signup item. The rule is to add some customization to this signup item. The detailed rules are described in the table below. |
| Action | - | Users can take some action here, such as moving this signup item up, moving this signup item down, or deleting this signup item. |

So far, the signup items that support configuration rules include `ID`, `Display name`, `Email` and `Agreement`.

| Item Name | Selectable Rules | Description |
| --- | --- | --- |
| ID | `Random` / `Incremental` | Select whether the user ID is randomly generated or incremented. |
| Display name | `None` / `Real name` / `First, last` | Choose the presentation of the display name. Choose `None` will show `Display name`. Choose `Real name` will show `Real name`. Choose `First, last` will show `First name` and `last name`. |
| Email | `Normal` / `No verification` | Select whether to verify the verification code of the mailbox. Choose `Normal` will verify the email code. Choose No verification will not |

| Item Name | Selectable Rules | Description |
| --- | --- | --- |
| | | verify the email code. |
| Agreement | `None` / `Signin` / `Signin (Default True)` | Select whether the user needs to confirm terms of use when logging in. Choose `None` to not display terms of use, and users can log in directly. Choose `Signin` to require users to confirm the terms before logging in. Choose `Signin (Default True)` to set the terms confirmed by default, and users can log in directly. |

> NOTE
>
> Here, for example, I want to setup my registration page bring a mailbox but do not require a mail verification code.

Firstly, I added some signup items necessary for registration, such as ID, Username, Password, Email.

And I selected the email row's rule item to `No verification`. As a result, the generated preview registration page can get the expected effect.

Casdoor

* Username:

* Password:

* Email:

Sign Up

Have account? sign in now

# Login UI Customization

You have created the application. Here will show you how to customize the login page UI of the application. In this guide we will create the following application login page:

Powered by Casdoor

Let's start!

## Part1: Add a background image

First, let's add a background image. The default background is white. It looks very simple.

- `Background URL` The background image url.

Choose the background image you like and fill the `Background URL`. The preview area will display the image, if you fill the valid url.

# Part2: Customize the login panel

Here's where you were at the end of the 1st part:

Now you need to add some css to make the panel look nice. You can copy the code below and paste it in the field `Form CSS`.

```
<style>
.login-panel{
    padding: 40px 30px 0 30px;
    border-radius: 10px;
    background-color: #ffffff;
    box-shadow: 0 0 30px 20px rgba(0, 0, 0, 0.20);
}
</style>
```

> **TIP**
>
> When you edit the `form CSS`, if the value is empty, the editor will show the default value. But it is not fill in the field. You need to copy the content and paste.

After filling the `form CSS`, don't forget to save the config at the bottom. Ok, let's see the effect.

# Part3: Select the Panel position

Now the login page is much prettier than it did at the beginning. We also provide three buttons for you to decide the position of the panel.

For example, select the **Right** button:

# Part4: Enable the side panel

You will see now how you can enable a side panel and customize the style.

First, select the button. In **enable side panel** mode, the panel will be in center.

Then edit the `Side panel HTML`, it decides what content will show in the side panel. Same as the `Form CSS`, we provide a default template. Just copy and paste.

```
<style>
  .left-model{
    text-align: center;
    padding: 30px;
    background-color: #8ca0ed;
    position: absolute;
    transform: none;
    width: 100%;
    height: 100%;
  }
  .side-logo{
    display: flex;
    align-items: center;
  }
  .side-logo span {
    font-family: Montserrat, sans-serif;
    font-weight: 900;
    font-size: 2.4rem;
    line-height: 1.3;
    margin-left: 16px;
    color: #404040;
  }
  .img{
    max-width: none;
    margin: 41px 0 13px;
  }
</style>
<div class="left-model">
  <span class="side-logo"> <img src="https://cdn.casbin.org/img/
casdoor-logo_1185x256.png" alt="Casdoor" style="width: 120px">
    <span>SSO</span>
  </span>
  <div class="img">
    <img src="https://cdn.casbin.org/img/casbin.svg"
alt="Casdoor"/>
  </div>
</div>
```

Let's see the effect. The side panel with a logo and image is shown, but the result was not satisfactory.

You need to modify and add some css in `form CSS`.

The final code is as follows.

```
<style>
  .login-panel{
    border-radius: 10px;
    background-color: #ffffff;
    box-shadow: 0 0 30px 20px rgba(0, 0, 0, 0.20);
}
  .login-form {
    padding: 30px;
  }
</style>
```

> INFO
>
> `.login-panel`, `.login-form` are the class names of div. They correspond to different areas of the page. For more details, you can check them through developer tools. After making sure the class names, you can customize the login page more flexibly by writing CSS here.

Finally, we get a beautiful login page.

# Review

OK, so let's sum it up: we have added a background image, customized the login panel style and enabled the side panel.

More introduction about application in Casdoor:

- Customize theme Customize the theme, including primary color, border radius.
- Signup Items Table
- Application Config

Thanks for reading!

# Permissions

## Overview

Using Casbin to manage users' access rights in organization

## Permission Configuration

Using exposed Casbin APIs to manage users' access rights in organization

## Exposed Casbin APIs

Using exposed Casbin APIs to manage users' access rights in organization

## Adapter

Config adapter and basic CRUD to policy

# Overview

## Introduction

All users associated with a single Casdoor organization are shared between the organization's applications and therefore have access to the applications. Sometimes you may want to restrict users' access to certain applications, or certain resources in a certain application. In this case, you can use `Permission` implemented by Casbin.

Before going further, you should have an understanding of how Casbin works and its related concepts, such as Model, Policy, and Adapter. In short, Model defines your permission policy structure, and how requests should match these permission policies and their effects. Policy is the description of your specific permission rules. After Casbin obtains Model and Policy information, it can enforce permission control on incoming requests. As an abstraction layer, Adapter shields the source of Policy for Casbin's executor, so that Policy can be stored everywhere, such as files or databases.

Back to the topic of permission configuration in Casdoor. In the Casdoor Web UI, you can add a Model for your organization in the `Model` configuration item, and a Policy for your organization in the `Permission` configuration item. With Casbin Online Editor, you can get Model and Policy files suitable for your usage scenarios. You can easily import the Model file into Casdoor through the Casdoor Web UI for use by the built-in Casbin. But for Policy (that is, the `Permission` configuration item in the Casdoor Web UI), some additional instructions are required here. Let us continue to mention later.

Just as your application needs to enforce permission control through the built-in

Casbin of Casdoor, as a built-in application, Casdoor also uses its Model and Policy to control the calling permissions of the API interface through Casbin. However, Casdoor can call Casbin from internal code, but external applications cannot. Therefore, Casdoor exposes an API for calling the built-in Casbin to external applications. We will show you the definitions of these API interfaces and how to use them later.

End of the chapter, we will use a practical example to show you how Casdoor cooperates with external applications for permission control.

Let's start!

# Permission Configuration

Let's explain each item in the Permission configuration page in turn.

- `Organization`: The name of the organization to which the policy belongs. An organization can have multiple permission policy files.
- `Name`: The permission policy name, which is globally unique in the organization. Used to identify the permissions policy file.
- `Display name`: Nothing important.
- `Model`: The name of the model file describing the permission policy structure and its matching patterns.
- `Adapter`: Attention! In the current version, this field describes the name of the database table that stores the permission policy, rather than the name of the adapter configured in the Adapter menu item in the Casdoor Web UI. Casdoor uses its own database to store permission policies being configured. If this field is empty, the permission policy will be stored in the `permission_rule` table, otherwise it will be stored in the specified database table. If the specified table name does not exist in the database used by Casdoor, it will be created automatically. We strongly recommend **specifying different adapters for different models**, as keeping all policies in the same table may cause conflicts.
- `Sub users`: Which users will be applied to this permission policy.
- `Sub roles`: If the RBAC model is used, which roles will be applied to the permission policy. This will add permission policies such as `g user role` for every user in this role.
- `Sub domains`: Which domains will be applied to this permission policy.
- `Resource type`: In fact, in the current version, Casdoor does not use this field

for external applications that want to authenticate. You can ignore it for now.

- `Resources`: This field describes the resources for which you wish to enforce permission control. Note, however, that the resources here are not those configured in the Resources menu item of the Casdoor Web UI. You can add any string you want here, such as a URL or a filename.
- `Actions`: This field describes actions to operate resources. Same as resource, it can be any string you want, such as Http Method or other natural language. But please note that Casdoor will convert all these strings to lowercase before storing. Additionally, Casdoor will apply all actions to each resource. You cannot specify that an action only take effect on certain resources.
- `Effect`: This option takes effect for Casdoor itself to control application access. If you want an external application to enforce permission controls using the interface Casdoor exposes, it won't do anything. You should describe the effect of pattern matching in the Model file.

As you can see, this configuration page is almost tailor-made for the `(sub, obj, act)` model.

# Exposed Casbin APIs

## Introduction

Let's assume that your application front-end has obtained the `access_token` of the logged-in user, and now wants to authenticate the user for some access. You cannot simply place the `access_token` to the HTTP request header to use these APIs, because Casdoor uses the `Authorization` field to check the access permission. Like any other APIs provided by Casdoor, the `Authorization` field consists of the application client id and secret, using the Basic HTTP Authentication Scheme. It looks like `Basic XXX`. For this reason, Casbin APIs should be called by the application backend server. Here are steps about how to do it.

1. The front end passes the `access_token` to the backend server through the HTTP request header.
2. The backend server gets the user id from the `access_token` which is also parameter `v0` in APIs described below.

As a note in advance, these interfaces are also pretty much designed (for now) for the `(sub, obj, act)` model. The `id` in the interface is the identity of the applied permission policy, which consists of the organization name and the permission policy name (ie `organization name/permission name`). `v0`, `v1` and `v2` in turn correspond to the policy structure described by the permission model, usually representing `sub`, `obj` and `act` respectively.

In addition to the API interface for requesting enforcement of permission control, Casdoor also provides other interfaces that help external applications obtain permission policy information, which is also listed here.

## Enforce

Request:

```
curl --location --request POST 'http://localhost:8000/api/enforce'
\
--header 'Content-Type: application/json' \
--header 'Authorization: Basic client_id_and_secret' \
--data-raw '{"id":"example-org/example-permission", "v0":"example-
org/example-user", "v1":"example-resource", "v2":"example-action"}'
```

Response:

```
true
```

## BatchEnforce

Request:

```
curl --location --request POST 'http://localhost:8000/api/batch-
enforce' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic client_id_and_secret' \
--data-raw '[{"id":"example-org/example-permission", "v0":"example-
org/example-user1", "v1":"example-resource", "v2":"example-
action"}, {"id":"example-org/example-permission", "v0":"example-
org/example-user2", "v1":"example-resource", "v2":"example-
action"}, {"id":"example-org/example-permission", "v0":"example-
org/example-user3", "v1":"example-resource", "v2":"example-
action"}]'
```

Response:

```
[
    true,
    true,
    false
]
```

## GetAllObjects

Request:

```
curl --location --request GET 'http://localhost:8000/api/get-all-objects' \
--header 'Authorization: Basic client_id_and_secret'
```

Response:

```
[
    "app-built-in"
]
```

## GetAllActions

Request:

```
curl --location --request GET 'http://localhost:8000/api/get-all-actions' \
--header 'Authorization: Basic client_id_and_secret'
```

Response:

```
[
    "read",
    "write",
    "admin"
]
```

## GetAllRoles

Request:

```
curl --location --request GET 'http://localhost:8000/api/get-all-roles' \
--header 'Authorization: Basic client_id_and_secret'
```

Response:

```
[
    "role_kcx66l"
]
```

# Adapter

Casdoor supports using the UI to connect the adapter and manage the policy rules. In Casbin, the policy storage is implemented as an adapter (aka middleware for Casbin). A Casbin user can use an adapter to load policy rules from a storage, or save policy rules to it.

## Adapter

- `type` : Adapter type. Now support database adapter.
- `Host`
- `Port`
- `User`
- `Password`
- `Database type` : Now support MySQL, PostgreSQL, SQL server, Oracle, SQLite 3.
- `Database` : The database name.
- `Table` : The table name. If the table does not exist, it will be created.
- `model` : You can select one model belonging to the organization of the adapter.

**Edit Adapter** Save **Save & Exit**

| | |
|---|---|
| Organization : | built-in |
| Name : | casdoor_adapter |
| Type : | Database |
| Host : | localhost |
| Port : | 3306 |
| User : | root |
| Password : | 123456 |
| Database type : | MySQL |
| Database : | casdoor |
| Table : | casbin_rule |
| Model : | casbin_rule |
| Policies : | Sync |

> **INFO**
>
> After fill all the fields, please don't forget to **save** the config. Then click the **sync** button to load the policy rules. The policy rules will be shown in the below table.

Policies : Sync

Add

| Rule Type | V0 | V1 | V2 | V3 | V4 | V5 | Option |
|---|---|---|---|---|---|---|---|
| p | built-in | * | * | * | * | * | |
| p | app | * | * | * | * | * | |
| p | * | * | POST | /api/signup | * | * | |
| p | * | * | POST | /api/get-email-and-phone | * | * | |
| p | * | * | POST | /api/login | * | * | |
| p | * | * | GET | /api/get-app-login | * | * | |
| p | * | * | POST | /api/logout | * | * | |
| p | * | * | GET | /api/logout | * | * | |
| p | * | * | GET | /api/get-account | * | * | |
| p | * | * | GET | /api/userinfo | * | * | |

< 1 2 3 4 5 >

Is enabled :

# Basic CURD

If you connect the adapter successfully, you can make basic CURD to the policy rules.

- Add

| Rule Type | V0 | V1 | V2 | V3 | V4 | V5 | Option |
|---|---|---|---|---|---|---|---|
| p | built-in | * | * | * | * | * | |
| p | * | * | POST | /api/signup | * | * | |
| p | * | * | POST | /api/get-email-and-phone | * | * | |
| p | * | * | POST | /api/login | * | * | |
| p | * | * | GET | /api/get-app-login | * | * | |
| p | * | * | POST | /api/logout | * | * | |
| p | * | * | GET | /api/logout | * | * | |
| p | * | * | GET | /api/get-account | * | * | |
| p | * | * | GET | /api/userinfo | * | * | |
| p | * | * | POST | /api/webhook | * | * | |

> **TIP**
>
> You can only add one policy at one time. The newly added policy is in the first row in the table, but actually, it will be saved in the last row. So next time you sync the policies, they will appear in the last row of the table.

- Edit

Model :
casbin_rule
Policies :
Sync
Add
Rule Type V0 V1 V2 V3 V4 V5 Option
p built-in * POST * * *
p app * * * * *
p * * POST /api/signup * *
p * * POST /api/get-email-and-phone * *
p * * POST /api/login * *
p * * GET /api/get-app-login * *
p * * POST /api/logout * *
p * * GET /api/logout * *
p * * GET /api/get-account * *
p * * GET /api/userinfo * *
1 2 3 4 5

- Delete

User :
root
Password :
123456
Database type :
MySQL
Database :
casdoor
Table :
casbin_rule
Model :
casbin_rule
Policies :
Sync
Add
Rule Type V0 V1 V2 V3 V4 V5 Option
p * * GET /api/get-default-application * *
p test * * * * *
1 2 3 4 5

# Providers

## Overview

Add third-party services to your application

## OAuth

19 items

## Email

Using email to complete authentication

## SMS

Using SMS to complete authentication

## 🗂️ Storage

3 items

## 🗂️ SAML

3 items

## 🗂️ Payment

2 items

## 🗂️ Captcha

6 items

# Overview

Casdoor uses providers to provide third-party services for the platform. In this chapter you will learn how to add providers for Casdoor.

## What we have

Now, we have 6 kinds of providers:

- OAuth providers

  Casdoor allows users to sign in through other OAuth applications. You can add GitHub, Google, QQ and many other OAuth applications to Casdoor. For more details, please see OAuth.

- SMS Providers

  Casdoor sends SMS to users when they want to verify their phone numbers. SMS providers are used to send SMS in Casdoor.

- Email Providers

  Email providers are similar to SMS providers.

- Storage Providers

  Casdoor allows users to store files using local file system or cloud oss services.

- Payment Provider

Casdoor can add payment providers, which will be used to add payment methods to products on the product page. Currently, the supported payment providers include Alipay, WeChat Pay, PayPal and GC.

- Captcha Provider

  Casdoor supports configurable captcha in user flows. Currently, the supported captcha providers include Default Captcha, reCAPTCHA, hCaptcha, Aliyun Captcha and Cloudflare Turnstile.

# How to config and use

## Scope

Providers have different scopes. The scope of the provider depends on the creator. Only the Administrator has the permission to add and config providers. There are two kinds of Administrator in Casdoor.

- Global Administrator: All users under the `built-in` organization and the user enable `IsGlobalAdmin`. The providers created by Global Administrator can be used by all applications.

- Organization Administrator: The user enable `IsAdmin`. The providers created by Organization Administrator can **only** be used by the applications under the organization. (Developing...)

## Add to application

Following under steps to add providers to your application. You still can **not** use the provider in your application before you add the provider to it.

1. Go to the application edit page and add a new provider row.

2. Select a provider you want to add to the application. Here will show all the providers that the application can use.

3. For **OAuth** and **Captcha** providers, you can config the usage. See OAuth and Captcha

Last, **save** the config. Then you can have a try to use the provider in your application.

# OAuth

## Overview

Add OAuth providers to your application

## Custom Provider

Add your own custom OAuth provider

## Twitter

Add Twitter OAuth provider to your application

## Weibo

Add Weibo OAuth provider to your application

## Wechat

Add Wechat OAuth provider to your application

## WeCom

Add WeCom OAuth provider to your application

## Tencent QQ

Add Tencent QQ OAuth provider to your application

## DingTalk

Add DingTalk OAuth provider to your application

## Steam

Add Steam OAuth provider to your application

## GitHub

Add Github OAuth provider to your application

## Gitee

Add Gitee OAuth provider to your application

## Linkedin

Add Linkedin OAuth provider to your application

## Facebook

Add Facebook OAuth provider to your application

## Google

Add Google OAuth provider to your application

## Baidu

Add Baidu OAuth provider to your application

## AD FS

Add AD FS as a third party service to complete authentication

## AzureAD

Add AzureAD as a third party service to complete authentication

## Infoflow

Add Infoflow OAuth provider to your application

## Okta

Add Okta OAuth provider to your application

# Overview

Casdoor can use other OAuth applications as a signin method.

Now, Casdoor supports many OAuth application providers. Icons of providers will be shown in login and signup pages after adding to Casdoor. Here are the providers Casdoor supports:

| Provider | Logo | Provider | Logo | Provider | Logo | Provider | Logo |
|---|---|---|---|---|---|---|---|
| Adfs | | Alipay | | Amazon | | Apple | |
| Auth0 | | AzureAD | | Baidu | | Battle.net | |
| Bilibili | | Bitbucket | | Box | | Casdoor | |
| Cloud Foundry | | Dailymotion | | Deezer | | DigitalOcean | |
| DingTalk | | Discord | | Douyin | | Dropbox | |
| Eve Online | | Facebook | | Fitbit | | Gitea | |
| Gitee | | GitHub | | GitLab | | Google | |
| Heroku | | InfluxCloud | | Infoflow | | Instagram | |
| Intercom | | Kakao | | Lark | | Lastfm | |

| Provider | Logo | Provider | Logo | Provider | Logo | Provider | Logo |
|---|---|---|---|---|---|---|---|
| Line | | LinkedIn | | Mailru | | Meetup | |
| MicrosoftOnline | | Naver | | Nextcloud | | Okta | |
| OneDrive | | Oura | | Patreon | | Paypal | |
| QQ | | SalesForce | | Shopify | | Slack | |
| SoundCloud | | Spotify | | Steam | | Strava | |
| Stripe | | TikTok | | Tumblr | | Twitch | |
| Twitter | | Typetalk | | Uber | | VK | |
| WeChat | | WeCom | | Weibo | | Wepay | |
| Xero | | Yahoo | | Yammer | | Yandex | |
| Zoom | | Email | | SMS | | | |

We will show you how to apply for a third-party service and add it to Casdoor.

# Apply to become a developer

Before this, there are some general concepts you need to understand.

- **RedirectUrl**, Redirect address after authentication, fill in your application address, such as `https://forum.casbin.com/`
- **Scope**, Permission granted to you by the user, such as basic profile, Email address and posts and others.
- **ClientId/AppId**, **ClientKey/AppSecret**, This is the most important information, and it is what you need to get after you apply for a developer account. You **can not share** the key/secret with anyone.

# Add an OAuth provider

1. Navigate to your Casdoor index page
2. Click `Providers` in the top bar
3. Click `Add`, then you can see a new provider in the list top
4. Click the new provider to modify it
5. Select `OAuth` in `Category`
6. Choose the OAuth provider you need in `Type`
7. Fill the most important information, `Client ID` and `Client Secret`

# Applied in application

1. Click `Applicaton` in the top bar and choose one application, edit
2. click provider add button, select the provider you just added
3. Modify the permissions of the provider, such as allowing registration, login, and unbinding
4. Done!

# Custom Provider

> **NOTE**
>
> Casdoor supports custom providers, but the custom providers must follow the standard process of 3-legged OAuth, and the return value of `Token URL` and `UserInfo URL` must follow the format specified by Casdoor.

First, go to the provider page of Casdoor, and create a new provider. Select “Custom” in the Type item. It can be seen that in addition to `Client ID` and `Client Secret`, you need to fill in `Auth URL`, `Scope`, `Token URL`, `UserInfo URL` and `Favicon`.

- `Auth URL` is the custom provider's OAuth login page address.

Suppose we fill in `https://door.casdoor.com/login/oauth/authorize` at the `Auth URL`, then when the user logs in with this custom provider, the browser will first jump to

```
https://door.casdoor.com/login/oauth/
authorize?client_id={ClientID}&redirect_uri=https://{your-casdoor-
hostname}/callback&state={State_generated_by_Casdoor}&response_type=code&scope={Scope}`
```

After authorization is completed, the custom provider should redirect to

```
https://{your-casdoor-hostname}/callback?code={code}
```

Then the code parameter in this URL will be recognized by Casdoor.

- `Scope` is the scope parameter carried when accessing the `Auth URL`, which is filled in according to the requirements of the custom provider.

- `Token URL` is the API address for obtaining accessToken.

  After obtaining the code in the previous step, Casdoor needs to use this code to get the accessToken.

  Suppose we fill in `https://door.casdoor.com/api/login/oauth/access_token` at the `Token URL`, then Casdoor will access the Token URL as follows

```
curl -X POST -u "{ClientID}:{ClientSecret}" --data-binary
"code={code}&grant_type=authorization_code&redirect_uri=https://{your-casdoor-
hostname}/callback" https://door.casdoor.com/api/login/oauth/access_token
```

  The custom provider should return at least the following

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6Ixxxxxxxxxxxxxx",
  "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6xxxxxxxxxxxxxx",
  "token_type": "Bearer",
  "expires_in": 10080,
  "scope": "openid profile email"
}
```

- `UserInfo URL` is the API address for obtaining user information by accessToken.

  Suppose we fill in `https://door.casdoor.com/api/userinfo` at the `UserInfo URL`, then Casdoor will access the UserInfo URL as follows

```
curl -X GET -H "Authorization: Bearer {accessToken}" https://door.casdoor.com/api/
userinfo
```

The custom provider should return at least the following

```
{
  "name": "admin",
  "preferred_username": "Admin",
  "email": "admin@example.com",
  "picture": "https://casbin.org/img/casbin.svg"
}
```

- `Favicon` is the logo URL of a custom provider.

  This logo will be displayed on Casdoor's login page along with other third-party login providers.

# Twitter

## Twitter(still working🚧)

> Twitter's application steps are somewhat troublesome, and the official restrictions are a bit strict, so it may be more difficult to apply for a developer account than other third-party platforms.

Visit Developer Portal, register if you don't have an account. Twitter needs to know what you are applying for a developer account for. You must fill in it carefully, otherwise it will not pass.

After the application is approved, create an application, fill in the callback address and other information, you need to do two things, which will be set is **Authentication settings** section.

- Manually turn on **3-legged OAuth**, for Sign in with Twitter, posting Tweets on behalf of other accounts and more.
- Enable **Request email address from users**, for getting user email address.

# Weibo

## Weibo✔

Weibo's developer account application is not difficult, but the speed is relatively slow. It takes about 2-3 days.

Visit Developer Website, filling in basic information and waiting for a long review...

After the review is approved, you can get the Client Id and Client Secret.

# Wechat

## WeChat✔

Visit WeChat developer platform, and register as a developer. After your web application or your mobile application is approved, then you get your App Id and App Secret.

The WeChat provider offers two different sets of keypairs:

- The first keypair (`Client ID`, `Client Secret`) is for `WeChat Open Platform`

`(⍰⍰⍰⍰⍰⍰)`, it's only for the PC login scenario. It can show QR code in PC browser and the user can use the WeChat APP in mobile phone to scan the code, so the PC browser will allow to sign in with WeChat.

- The second keypair (`Client ID 2`, `Client Secret 2`) is for `WeChat Media Platform (⍰⍰⍰⍰⍰⍰)`, it's only for the inside-WeChat-app login scenario. It allows the user to log in with WeChat built-in browser inside WeChat mobile APP, it will jump to your `WeChat Official Account (⍰⍰⍰⍰⍰)` to log in. It's notable that in the mobile scenario, WeChat itself doesn't support logging in outside of WeChat APP, like in other mobile browsers (H5) or APPs. This is a limitation of WeChat instead of Casdoor.

If you fill in the second keypair (`Client ID 2`, `Client Secret 2`) and enable the `Enable QR code` switch, when the user clicks on the WeChat button to log in, **Casdoor will first ask the user to follow the WeChat official account (⍰⍰⍰⍰⍰)**, then continue the login process. It's notable that this is only available in PC login scenario because a mobile phone cannot scan the QR code by itself. Casdoor will automatically skip this step when used in mobile scenario (aka the WeChat built-in browser inside WeChat mobile APP).

> TIP
>
> We recommend setting the two key sets at the same time, and linking your `WeChat Open Platform (⍰⍰⍰⍰⍰⍰)` account and `WeChat Media Platform (⍰⍰⍰⍰⍰⍰)` account together inside `WeChat Open Platform (⍰⍰⍰⍰⍰⍰)`. So a WeChat user logged-in through PC and mobile can be recognized as the same user in Casdoor.

> NOTE
>
> Due to the limitations of WeChat OAuth, there is currently no way to log in

> via WeChat in a 3rd-party mobile APP or in a mobile browser other than WeChat APP. The mobile login must happen inside WeChat APP for now.

For more detailed information, please visit WeChat Open Platform.

# WeCom

## Introduction

The WeCom provides the authorized login method of OAuth, which can obtain members' identity information from the webpage opened by the WeCom terminal, eliminating the need for login.

There are two different types of applications: **internal** applications and **third-party** applications.

## Basic

To configure a WeCom provider, the following table describes the required parameters.

Parameter Description:

| Parameter | Description |
| --- | --- |
| Sub type | Internal or Third-party |
| Method | Silent or Normal |
| Client ID | The enterprise CorpID |
| Client secret | The enterprise CorpSecret |

| Parameter | Description |
| --- | --- |
| Agent ID | Application agentid |

> INFO
>
> WeCom has two authorization methods. **Silent** authorization and **normal** authorization.
>
> **Silent authorization**: After the user clicks the link, the page is `redirect_URI? code=CODE&state=STATE`
>
> **Normal authorization**: After the user clicks the link, a middle page is displayed for the user to choose whether to authorize or not. After the user confirms the authorization, go to `redirect_uri?code=CODE&state=STATE`
>
> For more details, please see document.

# More

For more information about internal application, please see Internal application.

About Third-party application, please see Third-party application.

# Tencent QQ

## Tencent QQ✔

Visit authentication platform of QQ - Connect QQ.

First you need to apply to become a developer. After the review is approved, follow the instructions of the platform and get your Client Id and Client Secret.

# DingTalk

## DingTalk✔

Visit DingTalk developer platform and log in using your DingTalk account. After entering the platform, follow the instructions of the platform and you will get your Client Id and Client Secret.

For more detailed information, please visit DingTalk developer docs.

In addition, you need to add the following permissions to the dingtalk application:

# Steam

## Steam✔

Visit Steam WebAPI platform and log in using your Steam account, then apply for an API Key for your casdoor domain or ip, and finally fill in your API Key as Client Secret into Casdoor. (ClientID does not need to be filled, and your steam account needs to have games to apply for the API)

For more detailed information, please visit Steam WebAPI doc.

# GitHub

GitHub OAuth supports both web application flow and device flow. Please continue reading to obtain OAuth credential.

First, please visit GitHub developer settings to register a new GitHub App.

> CAUTION
>
> **Tricks:** We recommend that you use GitHub Apps to replace the OAuth Apps, because GitHub Apps can add multiple redirect uri, which can bring convenience when deploying test and production environments. GitHub official also recommend using GitHub Apps instead of OAuth Apps.

Then fill the **Github App name**, **Homepage URL**, **description** and **Callback URL**.

> **SET AUTHORIZATION CALLBACK URL CORRECTLY**
>
> In GitHub App config, the `Callback URL` must be **your Casdoor's callback url**, and the `Redirect URL` in Casdoor should be **your application callback url**
>
> For more details, please read App config

After registering your GitHub App, you can generate your `Client Secret` now!

Add a GitHub OAuth provider and fill the `Client ID` and `Client Secret` in your Casdoor

Now you can use GitHub as third party service to complete authentication.

# Gitee

To set up Gitee OAuth provider, please go to Gitee developer, if you have not created applications before, the gitee workbench would like this:

Then you can create your gitee app.

Fill in the **name**, **description**, **homepage** and **callback URL** and carefully choose the **permissions**.

> **SET AUTHORIZATION CALLBACK URL CORRECTLY**
>
> In Gitee OAuth config, the `authorization callback URL` must be **your Casdoor's callback url**, and the `Redirect URL` in Casdoor should be **your application callback url**
>
> For more details, please read App config

Then you can create you gitee app and get `Client ID` and `Client Secrets` now!

Add a Gitee OAuth provider and fill the `Client ID` and `Client Secrets` in your Casdoor.

Now you can use Gitee as third party service to complete authentication!

> **CAUTION**
>
> Since Casdoor needs to obtain the user's email, the email option must be checked, otherwise it will cause scope authorization errors.

# Linkedin

To set up Linkedin OAuth provider, please go to Linkedin developer to create a new app.

After filling in the form above and creating your app, you'll need to verify the Linkedin page associated with the app

> NOTE
>
> Only the company page administrator can verify your app, and give permission to your app

After your app is verified, you can continue:

Add Authorized redirect URLs for your app as **your Casdoor callback URL**.

SET AUTHORIZED REDIRECT URLS CORRECTLY

> In Linkedin OAuth config, the `authorized redirect URLs` must be **your Casdoor's callback url**, and the `Redirect URL` in Casdoor should be **your application callback url**
>
> For more details, please read App config

Then you can obtain your `Client ID` and `Client Secret`

Add a Linkedin OAuth provider and fill the `Client ID` and `Client Secret` in your Casdoor.

Now you can use Linkedin as third party service to complete authentication!

# Facebook

To set up Facebook OAuth provider, please go to Facebok developer to create a new app.

Select what kind of app you are going to create.

After filling in your name and contact email, you can enter facebook developer dashboard.

Then set up Facebook login:

Choose Web platform for this app:

After fill the website url, you can go to **Facebook Login** > **Settings**, and fill valid OAuth Redirect URIs

> **SET AUTHORIZED REDIRECT URLS CORRECTLY**
>
> In Facebook OAuth config, the `Valid OAuth Redirect URIs` must be **your Casdoor's callback url**, and the `Redirect URL` in Casdoor should be **your application callback url**
>
> For more details, please read App config

Basic app configuaration is almost done!

Switch mode from **In development** to **Live** in the top bar of dashboard

Then your `App ID` and `App secrets` can be used in Casdoor.

Add a Facebook OAuth provider and fill the `Client ID` and `Client Secrets` with `App ID` and `App Secrets` in your Casdoor.

Now you can use Facebook as third party service to complete authentication!

# Google

To set up Google OAuth provider, please go to Google API console and log in using your Google account.

Then navigate to **OAuth consent screen** tab to configure OAuth consent screen.

And register your Google app.

Then navigate to **Credential** tab.

Credentials + CREATE CREDENTIALS DELETE

Create credentials to access your enabled APIs. Learn more

## API Keys

| Name | Creation date |
|---|---|
| No API keys to display | |

## OAuth 2.0 Client IDs

| Name | Creation date |
|---|---|
| No OAuth clients to display | |

## Service Accounts

| Email | Name |
|---|---|
| No service accounts to display | |

And create Credential for your app:

## Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information. Learn more about OAuth client types.

Application type *

> **SET AUTHORIZED REDIRECT URIS CORRECTLY**
>
> In Google OAuth config, the `Authorized redirect URIs` must be **your Casdoor's callback url**, and the `Redirect URL` in Casdoor should be **your application callback url**
>
> For more details, please read App config

After create Client ID and obtain `Client ID` and `Client Secrets`

Add a Google OAuth provider and fill the `Client ID` and `Client Secret` in your Casdoor

Now you can use Google as third party service to complete authentication.

# Baidu

To set up Baidu OAuth provider, please read the Baidu documentation and follow their steps to complete the application creation.

After creating your app, the redirect url is set in the following position:

Add your Casdoor domain in the following position:

> **CAUTION**
>
> This part is very different from the actual situation in the documentation given by Baidu:
>
> 1. Adding the url to the callback url setting will most likely fail to validate the url and cause the login to fail, so we add our domain name to the domain setting.
> 2. Only one url or domain name can be added, which is very different from the documentation.

Then you can get `Client ID` and `Client Secrets` now!

Add a Baidu OAuth provider and fill the `Client ID` and `Client Secrets` in your Casdoor.

Now you can use Baidu as third party service to complete authentication!

> **GENERAL TROUBLESHOOTING**
>
> If you encounter a Baidu prompt that your redirect url is incorrect, here are some ways you might be able to fix it:
>
> 1. Add your domain name to the appropriate location, and then reset the Secret (Baidu reset Secret has a bug, it will prompt you an error, but after refreshing the page the Secret has been refreshed)
> 2. If the above methods do not solve the problem, we suggest you delete the application and create a new one, and set your domain name first.
>
> Another problem is that the user name returned by Baidu is masked, unlike its documentation which shows the user name and the displayed name, so we can currently only use the masked name as the user name.

# AD FS

To set up Active Directory Federation Service, please read the ADFS for the basic knowledge about the ADFS and AD FS Deployment Guide for how to set up a AD FS server. Do ensure that you have a fully operational AD FS server before you move on to further steps.

## Step1 Enabling Oauth via AD FS

See Enabling Oauth Confidential Clients with AD FS for details about creating an app step by step.

By the time you finish this step, you should have acquired clientId and clientSecret like this

AD FS
File Action View Window Help
AD FS
Service
Access Control Policies
Relying Party Trusts
Claims Provider Trusts
Application Groups
Application Groups
Name
Description
Actions
Application Groups
Group...
m Here
Add Application Group Wizard
Server application
Steps
Welcome
Server application
Configure Application Credentials
Configure Web API
Apply Access Control Policy
Configure Application Permissions
Summary
Complete
Name:
ADFSOAUTHCC - Server application
Client Identifier:
4e7a28c0-2e4b-403b-8251-b6fffb1c3b6a
Redirect URI:
Example: https://Contoso.com
Add
https://localhost:44323
Remove
Description:
< Previous
Next >
Cancel

In which the Client Identifier in first picture and the Secret in the second picture are supposed to be clientId and clientSecret in Oauth.

# Enable Casdoor AD FS Provider

Add a AD FS provider and fill the `Client ID` and `Client Secrets` in your Casdoor.

Home Organizations Users Roles Permissions Providers Applications Resources Tokens Recor

**Edit Provider** Save Save & Exit

Name ⓘ :

Display name ⓘ :

Category ⓘ : OAuth

Type ⓘ : Adfs

Client ID ⓘ

Client secret ⓘ

Domain ⓘ :

Provider URL ⓘ : https://openhome.alipay.com/platform/appManage.htm#/app/2021003111697088/overview

Save Save & Exit

# AzureAD

## Introduction

Azure Active Directory (Azure AD) simplifies application management by providing a single identity system for cloud and on-premise applications. Software as a Service (SaaS) applications, on-premise applications, and Line of Business (LOB) applications can be added to Azure AD. Users can then log in once for secure and seamless access to these applications, as well as Office 365 and other business applications provided by Microsoft.

## How to use?

The steps to register an app are shown below.

### step1. Register an application

First, Register an application. And choose an account type as needed. The demo station uses the type shown below.

# step2. Create a client secret

Create a `client secret` and save the value, it will be used later.

# step3. Add redirect URIs

Follow the example in the picture to add the redirect URIs for Casdoor.

# step4. Grant admin consent

The `user.read` API is open by default. You can add more scope according to your needs. Finally, remember to **grant admin consent**.

# step5. Create AzureAD provider in casdoor

The last step, add an AzureAD OAuth provider and fill the `Client ID` and `Client Secret` in your Casdoor.

**Edit Provider** Save Save & Exit

| | |
|---|---|
| Name : | provider_casdoor_azuread |
| Display name : | Casdoor AzureAD |
| Category : | OAuth |
| Type : | AzureAD |
| Client ID | 621cc0f0-055f-433f-9894-bfa1bfde169d |
| Client secret | *** |
| Provider URL : | https://portal.azure.com/#view/Microsoft_AAD_RegisteredApps/Appli |

Save Save & Exit

# Infoflow

To set up Infoflow OAuth provider, please go to Infoflow and log in using your Infoflow account.

First, please visit Infoflow Application.

And register your Infoflow app.

Then you can get `AgentID` now.

Then navigate to **Setting** tab, and create a new management group.

Add your structure to the address book permissions, and give it the permissions shown below. Also add the application you just created to the following location.

Add the sensitive interface permissions as shown below:

敏感接口权限 修改

| 接口名称 | 权限开放 |
| --- | --- |
| 获取部门成员 | ☑ |
| 获取部门列表 | ☐ |
| 获取成员信息 | ☑ |
| 获取标签成员 | ☐ |
| 维护通信录 | ☐ |
| 获取成员群组列表 | ☐ |
| 获取群组成员列表 | ☐ |
| 维护群组成员 | ☐ |
| 发送群组消息 | ☐ |
| 维护群组话题 | ☐ |
| 维护勋章 | ☐ |
| 通讯录搜索 | ☐ |

You will be able to see `CorpID` and `Secret` on the same page:

Add an Infoflow OAuth provider and fill the `Client ID` , `Client Secret` and `Agent ID` in your Casdoor.

Now you can use Infoflow as third party service to complete authentication!

# Okta

To set up Okta OIDC provider, first visit Okta Developer and sign up to get a developer account.

Navigate to **Applications > Applications** tab, click **Create App Integration**, select a **Sign-in method** of **OIDC - OpenID Connect**, and select an **Application type** of **Web Application**, then click **Next**.

Enter the **Sign-in redirect URIs** , such as `https://door.casdoor.com/callback`.

In the **Assignments** section, define the type of **Controlled access** for your app, then click **Save** to create the app integration.

Now you get `Client ID`, `Client secret`, and `Okta domain`.

Add a Okta OAuth provider in Casdoor dashboard, enter your `Client ID`, `Client secret`, and `Domain`.

> **SET DOMAIN CORRECTLY**
>
> Note that `Domain` is not just `Okta domain`, `/oauth2/default` should be appended to it.
>
> Visit Okta docs on authorization servers to get more details.

Now you can use Okta as third party service to complete authentication.

# Email

## Add an Email provider

1. Click `Add` to add a new provider.
2. Select `Email` in `Category`

3. Fill `Username`, `Password`, `Host`, `Port` of your smtp service.

4. Fill customized `Email Title` and `Email Content` and save.

# SMS

We use casdoor/go-sms-sender to send SMS for Casdoor. Now, `go-sms-sender` supports Aliyun, Tencent Cloud and Volc SMS APIs. You can raise an issue, or make a pull request if you want to support other SMS providers.

## Add a SMS provider

1. Click `Add` to add a new provider.
2. Select `SMS` in `Category`

3. Select your provider type (`Aliyun SMS`, `Tencent Cloud SMS` or `Volc Engine SMS`)

4. Get your information from Aliyun, Tencent Cloud or Volc Engine and fill them out.

# Example

> NOTE
>
> Here, I use Aliyun SMS service as an example

After I logged in my Aliyun workbench, click AccessKey to create ID and Secret.

By creating AccessKey, I get my AccessKey ID and AccessKey Secret:

Fill the `AccessKey ID` and `AccessKey Secret` to Casdoor `Client ID` and `Client Secret`.

# Storage

## Overview

Set up a storage provider for upload files in Casdoor

## Azure Blob

Using Azure Blob as a storage provider for Casdoor

## Aliyun OSS

Using Aliyun OSS as a storage provider for Casdoor

# Overview

If you need to use file storage services such as `avatar upload`, you need to set up a storage provider and apply it in your `application`.

Casdoor supports two types of storage, **Local** and **Cloud**. In this chapter you will learn how to add a storage provider to use these services.

## Item

- `Client ID`
- `Client secret`
- `Endpoint`
- `Path perfix`: Path prefix for the file location.

> INFO
>
> Default `Path perfix` is `/`. For example, when the `Path perfix` is empty, a default file path:
>
> ```
> https://cdn.casbin.com/casdoor/avatar.png
> ```
>
> You can fill it with `abcd/xxxx`, and then you can store your avatar in:
>
> ```
> https://cdn.casbin.com/abcd/xxxx/casdoor/avatar.png
> ```

- `Bucket`
- `Domain`: The custom domain name of CDN for your cloud storage.

# Local

With **Local** type, the only item that you need to configure is `Domain` field. Please follow the format:

```
Domain/images
```

For example, `http://127.0.0.1:7001/images`, `http://door.casbin.org/images` are all allowed.

But `127.0.0.1:7001/images` is wrong.

The `Client ID`, `Client secret`, `Endpoint` and `Bucket` field are no longer needed, you can **fill in it at will**, and it **cannot** be empty.

# Cloud-Based

Currently, we support **AWS S3**, **Aliyun OSS**, **Tencent Cloud COS**, **MinIO** and **Azure Blob** cloud vendors, and are adding more Cloud storage services.

Fill in the corresponding fields with `Client ID`, `Client secret`, `Endpoint` and `Bucket` obtained from your cloud vendor console.

> TIP
>
> With `Non-local` type, you probably don't need the `Domain` field, which is used for custom domain.

# Azure Blob

> NOTE
>
> This is an example of **Azure Blob**

- You must have an Azure storage account.

## Step1. Select Azure Blob

Select the Azure Blob as the storage type.

## Step2. Fill the necessary information in Casdoor

There are three required fields. `Client ID`, `Client secret`, `Bucket`. The relationship corresponding to the Azure Blob account is as follows:

| Name | Name in Azure | is required |
| --- | --- | --- |
| Client ID | AccountName | required |
| Client secret | AccountKey | required |
| Bucket | ContainerName | required |
| Domain | DomainName | |

- AccountName

  `AccountName` is your AccountName.

- AccountKey

  The `AccountKey` is your key to access Azure API.

> NOTE
>
> You can obtain your account key from the Azure Portal under the "Access Keys" section on the left-hand pane of your storage account.

- ContainerName

  You first need to create a container. There is a default container called 'default'.

- Domain

The custom domain name in your Azure CDN.

# Step3. Save your configuration

The final result is as follows:

| | |
|---|---|
| Name : | Provider_azure |
| Display name : | Provider_azure |
| Category : | Storage |
| Type : | Azure Blob |
| Client ID | casbin |
| Client secret | *** |
| Endpoint : | |
| Endpoint (Intranet) : | |
| Bucket : | default |
| Domain : | https://cdn.casploy.com/ |
| Provider URL : | https://github.com/organizations/xxx/settings/applications/1234567 |

Then you can use Azure Blob Storage services in your application.

# Aliyun OSS

> NOTE
>
> This is an example of **Aliyun OSS**.

The AccessKey is your key to access Aliyun API, with full account permissions.

So created AccessKey in Aliyun workbench.

Then create OSS service:

Fill the necessary information in Casdoor and save:

| | |
|---|---|
| Name : | provider_storage_aliyun_oss |
| Display name : | Storage Aliyun OSS |
| Category : | Storage |
| Type : | Aliyun OSS |
| Client ID | LTAIxFoNpNAnPoiT |
| Client secret | *** |
| Endpoint : | oss-cn-beijing.aliyuncs.com |
| Endpoint (Intranet) : | oss-cn-beijing-internal.aliyuncs.com |
| Bucket : | casbin |
| Domain : | https://cdn.casbin.com/casdoor/ |
| Provider URL : | https://oss.console.aliyun.com/bucket/oss-cn-beijing/casbin/object |

Then you can use Aliyun cloud storage services in your application.

# SAML

## Overview

Using identities from external identity providers that support SAML 2.0

## Aliyun IDaaS

Using Aliyun IDaaS to authenticate users

## Keycloak

Using Keycloak to authenticate users

# Overview

Casdoor can be configured to support user login to UI using identities from external identity providers that support SAML 2.0. In such a configuration, Casdoor can never store any credentials for the users.

Now, Casdoor supports many SAML application providers. Icons of providers will be shown in login page after adding to Casdoor. Here are the providers Casdoor supports:

| Aliyun IDaaS | Keycloak |
| --- | --- |
| | |
| ✅ | ✅ |

## Terms

- Identity Provider (IDP) - The service that stores the identity database and provides identity and authentication services to Casdoor.
- Service Provider (SP) - The service providing resources to the end user, in this case, the Casdoor deployment.
- Assertion Consumer Service (ACS) - The consumer of SAML assertions generated by the Identity Provider.

# How SAML integration works

When using SAML SSO, users log into the Casdoor via the identity provider without ever passing credentials to Casdoor. The progress is shown in the following diagram.

# Aliyun IDaaS

## Create SAML application in Aliyun IDaaS

Login to the Aliyun management console, search and go to the Application Indentity Service (IDentity-as-a-Service, IDaaS).

Click **EIAM Instance List** and open the free version.

An instance will be created and run automatically after opening. Click on the instance name or the **Manage** button to enter the IDaaS management console.

After entering the IDaaS management console, click **Add Application**, search for **SAML**, and click **Add Application**.

Click **Add SigningKey**.

Fill in all required information and submit.

Select the added SigningKey.

Fill in all the required information below and submit.

- IDP IdentityId: Keep the same as Issuer URL in Casdoor.
- SP Entity ID & SP ACS URL(SSO Location): Now fill in whatever you want. After completing the configuration of Casdoor, you need to come to modify.
- Assertion Attribute: Directly fill in as username.
- Account Association Mode: Account Association

# Account authorization & association

After the application is successfully added, an authorization prompt will pop up. Do not authorize it now, add an account and then authorize it.

Go to **Organizations and Groups** and click on **New Account**.

Fill in all required information and submit.

Go to **Application Authorization**, select the accounts you want to authorize and click **Save**.

Go to the **Application List**, click **View application sub-accounts**, and then click **Add account association**.

Fill in the primary and sub accounts that need to be associated and click **Save**.

The primary account exists in IDaaS, and the sub account is the ID of the user in Casdoor.

# Export IDaaS Metadata

Go to the **Application List**, click **View Application Details** and click **Export IDaaS SAML Metadadta**.

# Configure in Casdoor

Create a new provider in Casdoor.

Select category as **SAML**, type as **Aliyun IdaaS**. Copy the content of metadata and paste it to the **Metadata** input. The values of **Endpoint**, **IdP** and **Issuer URL** will be generated automatically after clicking the **Parse** button.

Copy the **SP ACS URL** and the **SP Entity ID** and click the **Save** button.

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# Modify SAML application in Aliyun IDaaS

Disable the application and then click **Modify Application**.

Fill in **SP Entity ID** and **SP ACS URL(SSO Location)** with the content copied in Casdoor. Submit and enable application.

# Validate the effect

Go to the application you just configured and you can find that there is an icon in the login page.

Click the icon and jump to the Aliyun IDaaS login page, and then successfully login to the Casdoor after authentication.

casbin
username, Email or phone
Password
Auto sign in
Forgot password?
Sign In
Sign in with code
No account? sign up now

# Keycloak

The JBoss KeyCloak system is a widely used and open-source identity management system that supports integration with applications via SAML and OpenID Connect. It also can operate as an identity broker between other providers such as LDAP or other SAML providers and applications that support SAML or OpenID Connect.

The following is an example of how to configure a new client entry in KeyCloak and configure Casdoor to use it to permit UI login by KeyCloak users that are granted access via KeyCloak configuration.

## Configure Keycloak

Some config choices and assumptions specifically for this example:

- Let’s assume that you are running Casdoor as dev mode locally. Casdoor UI is available at: `http://localhost:7001` and server is available at `http://localhost:8000`. Replace with the appropriate url as needed.
- Let's assume that you are running Keycloak locally. Keycloak UI is available at: `http://localhost:8080/auth`.
- Based on that, the SP ACS URL for this deployment will be: `http://localhost:8000/api/acs`.
- Our SP Entity ID will use the same url: `http://localhost:8000/api/acs`.

Use the default realm or create a new realm.

# Add a client entry in Keycloak

> **INFO**
>
> See more details about Keycloak Clients in Keycloak documentation.

Click **Clients** in the menu and then click **Create** to go to the **Add Client** page. Fill in as below.

- **Client ID:** `http://localhost:8000/api/acs` - This will be the SP Entity ID used in the Casdoor configuration later.
- **Client Protocol:** `saml`.
- **Client SAML Endpoint:** `http://localhost:8000/api/acs`. - This URL is where you want the Keycloak server to send SAML requests and responses.

Generally, applications have one URL for processing SAML requests. Multiple URLs can be set in the Settings tab of the client.

Click **Save**. This action creates the client and brings you to the **Settings** tab.

The following list part of settings:

1. **Name** - `Casdoor`. This is only used to display a friendly name to Keycloak users in the KeyCloak UI. You can use any name you like.
2. **Enabled** - Select on.
3. **Include Authn Statement** - Select on.
4. **Sign Documents** - Select on.
5. **Sign Assertions** - Select off.
6. **Encrypt Assertions** - Select off.
7. **Client Signature Required** - Select off.
8. **Force Name ID Format** - Select on.
9. **Name ID Format** - Select username.
10. **Valid Redirect URIs** - Add `http://localhost:8000/api/acs`.
11. **Master SAML Processing URL** - `http://localhost:8000/api/acs`.
12. Fine Grain SAML Endpoint Configuration
    i. **Assertion Consumer Service POST Binding URL** -

`http://localhost:8000/api/acs`.

ii. Assertion Consumer Service Redirect Binding URL - `http://localhost:8000/api/acs`.

Save the configuration.

KEYCLOAK
Admin
Casdoor
Configure
Realm Settings
Clients
Client Scopes
Roles
Identity Providers
User Federation
Authentication
Manage
Groups
Users
Sessions
Events
Import
Export
Clients > http://localhost:8000/api/acs
Http://localhost:8000/api/acs
Settings
Roles
Client Scopes
Mappers
Scope
Sessions
Offline Access
Clustering
Installation
Client ID
http://localhost:8000/api/acs
Name
Casdoor
Description
Enabled
ON
Always Display in Console
OFF
Consent Required
OFF
Login Theme
Client Protocol
saml
Include AuthnStatement
ON
Include OneTimeUse Condition
OFF
Force Artifact Binding
OFF
Sign Documents
ON
Optimize REDIRECT signing key lookup
OFF
Sign Assertions
OFF
Signature Algorithm
RSA_SHA256
SAML Signature Key Name
KEY_ID
Canonicalization Method
EXCLUSIVE
Encrypt Assertions
OFF
Client Signature Required
OFF
Force POST Binding
OFF
Front Channel Logout
ON
Force Name ID Format
ON
Name ID Format
username
Root URL
Valid Redirect URIs
http://localhost:8000/api/acs
Base URL
Master SAML Processing URL
http://localhost:8000/api/acs
IDP Initiated SSO URL Name
IDP Initiated SSO Relay State
Fine Grain SAML Endpoint Configuration
Assertion Consumer Service POST Binding URL
http://localhost:8000/api/acs
Assertion Consumer Service Redirect Binding URL
http://localhost:8000/api/acs
Logout Service POST Binding URL
Logout Service Redirect Binding URL
Logout Service ARTIFACT Binding URL
Artifact Binding URL
Artifact Resolution Service
Advanced Settings
Authentication Flow Overrides
Save
Cancel

TIP

> If you want to sign authn request, you need to enable the **Client Signature Required** option and upload the certificate generated by yourself. The private key and certificate using in Casdoor, `token_jwt_key.key` and `token_jwt_key.pem` are located in the **object** directory. In Keycloak, you need to click the **Keys** tab, click **Import** button, select **Archive Format** as **Certificate PEM** and upload the certificate.

Click **Installation** tab.

For the Keycloak <= 5.0.0, select Format Option - **SAML Metadata IDPSSODescriptor** and copy the metadata.

For Keycloak 6.0.0+, select Format Option - **Mod Auth Mellon files** and click **Download**. Unzip the downloaded.zip, locate `idp-metadata.xml` and copy the metadata.

# Configure in Casdoor

Create a new provider in Casdoor.

Select category as **SAML**, type as **Keycloak**. Copy the content of metadata and paste it to the **Metadata** input. The values of **Endpoint**, **IdP** and **Issuer URL** will be generated automatically after clicking the **Parse** button. Finally click the button **Save**.

> **TIP**
>
> If you enable the **Client Signature Required** option in Keycloak and upload the certificate, please enable the **Sign request** option in Casdoor.

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# Validate the effect

Go to the application you just configured and you can find that there is a Keycloak icon in the login page.

Click the icon and jump to the Keycloak login page, and then successfully login to the Casdoor after authentication.

# Payment

## Alipay

## WeChatPay

Add Wechat OAuth provider to your application

# Alipay

# WeChatPay

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation. Please deploy your Casdoor instance in **production mode**.

After a successful deployment, you need to ensure:

- Open your favorite browser and visit **http://localhost:8000**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2. Configure payment callback notification address

Before that, please log in to the WeChat Pay Merchant Platform. In order for Casdoor to receive payment result notifications, you need to set a callback notification address in the WeChat Pay Merchant Platform.

- In Merchant Dashboard, go to `Development Configuration` → `Payment Configuration`.
- Find the `Callback Notification Address` setting, and click the `Modify`

button.

- Fill in the payment callback notification address of the Casdoor instance. For example:

```
https://your-casdoor-instance.com/api/wechat-payment-callback
```

# Step3. Configure API Security

## Get APIv3 key

To ensure the security of API calls, you need to configure API security settings in the WeChat Pay Merchant Platform.

Log in the WeChat merchant platform → `Account center` → `Account settings` → `API security` → `APIv3 key` → `set up`

## Get the serial number of the merchant certificate

It is also necessary to obtain the serial number of the merchant certificate. The following are the steps to obtain it.

Log in the WeChat merchant platform → `Account center` → `Account settings` → `API security` → `API certificate management` → copy the serial number.

You can refer to the WeChat payment merchant ID query guide , APIv3 key settings and How to view the certificate serial number for help.

# Step4. Add payment provider

## Select the WeChat Pay as the Payment type.

## Get Payment URL

Log in the WeChat merchant platform → `Commodity centered` → `Development arrangement` → `payment arrangements` → `Payment URL` → `copy`

## Fill the necessary information in Casdoor

There are four required fields, `Client ID`, `Client secret`, `appId`, `Provider URL`. The relationship corresponding to the Azure Blob account is as follows:

| Name | Name in WeChat Pay | is required |
|---|---|---|
| Client ID | merchant ID | required |
| Client secret | APIv3 key | required |
| appId | appId | required |
| Provider URL | Payment URL | required |

The acquisition of `merchant ID` and `APIv3 key` is as mentioned before. For appId, see here for more help.

# Step5. Add WeChat Cert

In this step, you need two required fields, `serial number`, `private key`.

How to obtain the serial number has been stated in the third step.

To get `private key`, click here for help.

# Step6. Add WeChat Payment Provider in Products

Final effect:

# What's more

You can explore the following projects/docs to learn more about WeChat Pay.

- wechatpay-apiv3-go-sdk
- Wechat payment development document

# Captcha

## Overview

Add a captcha to your application

## Default

Using Casdoor default captcha in your application

## reCAPTCHA

Add reCAPTCHA to your application

## hCaptcha

Add hCaptcha to your application

## Aliyun Captcha

Add Aliyun Captcha to your application

## Cloudflare Turnstile

Add Cloudflare Turnstile to your application

# Overview

Casdoor can be configured to support different captchas to check whether the operation is made by human. If you add a captcha provider and applied it in the application, when the user logins, registers or forgets password and needs to send a code, then a captcha check dialog will appear to check whether the operation is made by human.

Now, Casdoor supports many captcha providers. Here are the providers Casdoor supporting:

| Default | reCAPTCHA | hCaptcha | Aliyun Captcha | Cloudflare Turnstile |
|---|---|---|---|---|
| | | | | |
| ✅ | ✅ | ✅ | ✅ | ✅ |

We will show you how to apply a captcha and add it to Casdoor.

## Add a captcha provider

1. Navigate to your Casdoor index page
2. Click `Providers` in the top bar
3. Click `Add`, then you can see a new provider in the list top
4. Click the new provider to modify it
5. Select `Captcha` in `Category`

6. Choose the Captcha provider you need in `Type`
7. Fill the most important information, different captcha providers have different information that needs to be filled in

# Applied in application

1. Click `Applicaton` in the top bar and choose one application to edit.
2. Click provider add button, and select the provider you just added.
3. Done!

# Default

Default captcha implements generation and verification of image. A default captcha image is the sequence of digits 0-9 with the defined length(5).

## Configure in Casdoor

Create a new provider in Casdoor.

Select category as **Captcha** , type as **Default** .

And you can click **Preview** button to preview the style of this captcha.

# Applied in application

Edit the application you want to configure in Casdoor. Select the provider just added. There are three kinds of rules:

- `Always` Always turned on human-machine verification when login.
- `None` Never require human-machine verification, the accout will be blocked when it attempted to login into the same application with wrong password for the 5st time within 15 minutes. And it will be unblocked after 15 minutes.
- `Dynamic` After 5 failed login attempts, the account will not be blocked but instead, human-machine verification will be required.

We also provide a demo video to demonstrate the differences in rules, which we hope will be helpful to you

# reCAPTCHA

reCAPTCHA is provided by Google. And we use reCAPTCHA v2 Checkbox . You can see more details from this link.

## Create an API key pair

To start using reCAPTCHA, you need to sign up for an API key pair for your site. The key pair consists of a site key and secret key. The site key is used to invoke reCAPTCHA service on your site or mobile application. The secret key authorizes communication between your application backend and the reCAPTCHA server to verify the user's response.

First, choose the type of reCAPTCHA and then fill in authorized domains or package names. After you have accepted the terms of service, click **Register** to get a new API key pair.

Then you can get a site key and a secret key.

# Configure in Casdoor

Create a new provider in Casdoor.

Select category as **Captcha** , type as **reCAPTCHA** . And you need to fulfill the site key and the secret key which is created by last step.

And you can click **Preview** button to preview the style of this captcha.

# Applied in application

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# hCaptcha

hCaptcha is a captcha service provider which is similar to reCAPTCHA. You can see more details from this link.

## Create an API key pair

To start using hCaptcha, you need to sign up for an API key pair for your site. You can find your site key on your profile page.

Then you can get a site key and a secret key.

## Configure in Casdoor

Create a new provider in Casdoor.

Select category as **Captcha** , type as **hCaptcha** . And you need to fulfill the site key and the secret key which is created by last step.

And you can click **Preview** botton to preview the style of this captcha.

# Applied in application

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# Aliyun Captcha

Aliyun Captcha is a captcha service provided by Aliyun. It includes two ways to verify captcha: `Sliding Validation` and `Intelligent Validation`. You can see more details from this link.

## Add Captcha configuration in Aliyun

Login to the Aliyun management console, search and go to the Captcha Service. And click **Confirm Open** to enable Captcha Service.

After entering the captcha agement console, click **Add configuration**.

Fill in all required information and submit.

Then you can see your `Scene` and `App key` in your console.

And `Access key`, `Secret access key` is in your profile.

# Configure in Casdoor

Create a new provider in Casdoor.

Select category as **Captcha** , type as **hCaptcha** . Then select sub type: `Sliding Validation` or `Intelligent Validation`. And you need to fulfill the `Access key`, `Secret access key`, `Scene` and `App key` which are created by last step.

And you can click **Preview** button to preview the style of this captcha.

The following image is `Sliding Validation` preview:

The following image is `Intelligent Validation` preview:

# Applied in application

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# Cloudflare Turnstile

Cloudflare Turnstile is a captcha service provided by Cloudflare, which is a user-friendly, privacy preserving alternative to captcha. You can see more details from Turnstile Docs.

## Create an API key pair

To start using Cloudflare Turnstile, you need to Create a Cloudflare account, navigate to the `Turnstile` tab on the navigation bar, and get the Site Key and Secret Key.

First, add a name for the widget to identify it in the future and enter your website's hostname. Then choose the widget type. `Managed` is recommended. Click **Create**.

Then you can get a site key and a secret key.

# Configure in Casdoor

Create a new provider in Casdoor.

Select category as **Captcha** , type as **Cloudflare Turnstile** . And you need to fulfill the site key and the secret key which is created by last step.

And you can click the **Preview** button to preview the style of this captcha.

# Applied in application

Edit the application you want to configure in Casdoor. Select the provider just added and click the button **Save**.

# Resources

## Overview

Upload resources in Casdoor

# Overview

You can upload resources in casdoor. Before upload resources, you need to configure a storage provider. Please see Storage Provider

You have now configured at least one **storage provider** and added that provider to your **application**.

All right! Let's see an example of how to **upload** and **delete** resources.

## Upload Resources

Users can upload resources such as files and images to the previously configured cloud storage.

# Delete Resources

If you no longer need the resource, you can choose to delete it by clicking the "Delete" button.

# Products

## Products

Add products that you want to sell

## Payment

View the transaction information of the products in Payment

# Products

You can add the product (or service) you want to sell. The following will tell you how to add a product.

## Configuring Products Attributes

First, you need to understand the basic properties of the product: `Tag` `Detail` `Currency` `Price` `Quantity` `Sold`

## Payment Provider

Of course, in addition to setting these properties, you also need to add payment

providers to the product, and multiple payment providers can be added to a product.

To learn how to configure a payment provider, see Payment Provider

Finally, fill in the **Return URL**. This is the url to jump from the payment provider page when the payment is completed.

# Preview the Product

You're done. See the review and save:

# Payment

After the payment is successful, you can see the transaction information of the products in **Payment**, such as organization, user, purchase time, product name, etc.

## Invoice

You can enter the edit screen to issue an invoice

Fill in invoice information, invoice types are `individual` and `organization`.

Finally, click the "**issue invoice**" button.

# Pricing

## Overview

Casdoor pricing overview

## Plan

Casdoor plan overview

## Pricing

Casdoor pricing overview

## Subscription

Casdoor subscription overview

# Overview

Casdoor can be used as subscription management system via `plan`, `pricing` and `subscription`.

You can choose which plans to include in your price list like on pictures below:

Each `pricing list` belong to `application` and has own url that allow sign up new users with selected plan.

Getneral user flow looks like:

- Get `pricing` url
- Choose `plan`

- Signup to aplication
- `Subscription` will be auto created

# Plan

`Plan` - describe list of application's features with own name and price.

Plan features depends on Casdoor `role` with set of `permissions`.

That allow to describe plan's features independ on naming and price. For example: plan may has diffrent prices depends on county or date.

Picture below describes relation between Plan and role.

Plan

- Display Name
- Price per month

...

Role

permission 1

permission 2

...

permission N

# Plan properties

Every plan has these properties:

- `Owner`
- `Name`
- `CreatedTime`
- `DisplayName`
- `IsEnabled`
- `PricePerMonth`
- `PricePerYear`
- `Role`

# Pricing

`Pricing` contains one or more `plan` and provide ability to sing up to application with selected plan.

General flow might looks like on picture below:

## Pricing properties

Every Pricing has these properties:

- `Owner`
- `Name`
- `CreatedTime`
- `DisplayName`
- `Description`
- `Plans` - Array of plans
- `IsEnabled`

- `Has trial` - any payments will not be reqiured for sign up in case of true
- `TrialDuration` - impact on subscription end days
- `Application`
- `Submitter`
- `Approver`
- `ApproveTime`
- `State`

# Subscription

`Subscription` - helps to manage user's selected plan that make easy to control application's features access.

> TIP
>
> Since each plan based on `casdoor role` you can assign plan's role to user and use enforce API for permission checking.

`Subscription` can be created in thee ways:

- Manually by admin
- After sign up from pricing page
- Via API.

## Subscription properties

Every Subscription has these properties:

- `Owner`
- `Name`
- `CreatedTime`
- `DisplayName`
- `Duration`
- `Description`
- `Plan`
- `StartDate`

- `EndDate`
- `User`
- `IsEnabled`
- `Submitter`
- `Approver`
- `ApproveTime`
- `State`

# Users

## Overview

Manage users in Casdoor

## Secure your account with MFA

Config adapter and basic CRUD to policy

## Roles

the user's roles

## Permissions

the user's permissions

# Overview

## User properties

As an authentication platform, Casdoor is able to manage users. Every user has these properties:

- `Owner` Owner organization of the user
- `Name` User name, unique
- `CreatedTime`
- `UpdatedTime`
- `Id` Unique for every user
- `Type`
- `Password`
- `PasswordSalt`
- `DisplayName` Shown in UI
- `FirstName`
- `LastName`
- `Avatar` A link to user's avatar
- `PermanentAvatar`
- `Email`
- `Phone`
- `Location`
- `Address`
- `Affiliation`

- `Title`
- `IdCardType`
- `IdCard`
- `Homepage`
- `Bio`
- `Tag`
- `Region`
- `Language`
- `Gender`
- `Birthday`
- `Education`
- `Score`
- `Karma`
- `Ranking`
- `IsDefaultAvatar`
- `IsOnline`
- `IsAdmin` Is the user the admin of his organization
- `IsGlobalAdmin` Does the user have the permission to manage the Casdoor
- `IsForbidden`
- `IsDeleted`
- `SignupApplication`
- `Hash`
- `PreHash`
- `CreatedIp`
- `LastSigninTime`
- `LastSigninIp`

- `Roles` Array of the user's roles
- `Permissions` Array of the user's permissions

unique Id of the platform:

- `Github`
- `Google`
- `QQ`
- `WeChat`
- `Facebook`
- `DingTalk`
- `Weibo`
- `Gitee`
- `LinkedIn`
- `Wecom`
- `Lark`
- `Gitlab`
- `Adfs`
- `Baidu`
- `Casdoor`
- `Infoflow`
- `Apple`
- `AzureAD`
- `Slack`
- `Steam`
- `Ldap`
- `Properties` This is a string → string map, stored all other properties may need.

# Import users from XLSX file

You can add new users or update existing Casdoor users by uploading a XLSX file of user information.

In the Admin Console, go to Users and click **Upload(.xlsx)** button.

Select your XLSX file and click Open, the users will be imported.

We provide a template XLSX file named `user_test.xlsx` in the `xlsx` folder. The template includes 5 users for test and headers for some required user properties.

Home Organizations Users Roles Permissions Providers Applications Users uploaded successfully, refreshing the page Syncers Certs Swagger Admin

Users Add Upload (.xlsx)

user_test.xlsx

| Organization | Application | Name | Created time | Display name | Avatar | Email | Phone | Affiliation | Country/Region | Tag | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|
| built-in | app-built-in | tesla | 2022-03-20 20:49:03 | Nikola Tesla | | 9v73hn@example.com | 40738134827 | Example Inc. | United States of America | scier | Edit Delete |
| built-in | app-built-in | gauss | 2022-03-20 20:48:33 | Carl Friedrich Gauss | | vqdsan@example.com | 98621482844 | Example Inc. | Germany | matl | Edit Delete |
| built-in | app-built-in | galieleo | 2022-03-20 20:47:58 | Galileo Galilei | | 8p4f38@example.com | 22596937332 | Example Inc. | Italy | scier | Edit Delete |
| built-in | app-built-in | euler | 2022-03-20 20:47:08 | Leonhard Euler | | 3dzw4j@example.com | 74409642681 | Example Inc. | Switzerland | matl | Edit Delete |
| built-in | app-built-in | einstein | 2022-03-20 20:46:29 | Albert Einstein | | z6mive@example.com | 60062541396 | Example Inc. | Germany | scier | Edit Delete |
| built-in | app-built-in | admin | 2022-03-20 20:07:23 | Admin | | admin@example.com | 12345678910 | Example Inc. | | staff | Edit Delete |

6 in total 1 10 / page

Made with ♥ by **Casdoor**

# Secure your account with MFA

## About multi-factor authentication

MFA (Multi-Factor Authentication) is a security measure that can improve the security of users and systems. It requires users to provide two or more factors of authentication to verify their identity when logging in or performing sensitive operations.

For Casdoor, the second form of authentication is a code that's sent as a text message or email. After you enable MFA, Casdoor generates an authentication code any time someone attempts to sign in your account. The only way someone can sign in your account is if they know both your password and have access to the authentication code.

## Config MFA

1. In the Use page, click the "add" button.

2. After added a new line, click the "setup" button.

3. Type your password and click "Next Step".

# Configuring multi-factor authentication using text messages or email

- Select your country code and type your mobile phone number. When your information is correct, click "Send Code".
- If you want to use email, type your email address. Then click "Send Code".

You'll receive a text message or email with a security code. On Casdoor, type the code into the field "Enter your code" and click "Next Step".

Above "Enable" button, copy your recovery codes and save to your device. Save them to a secure location because your recovery codes can help you get back into your account if you lose access.

# Changing your preferred MFA method

You can add multiple MFA methods. Only the preferred method will be used when you sign in.

If you want to set a preferred MFA method, click the "Set preferred" button.

A "Preferred" label is displayed on your preferred method.

# Roles

Each user may have multiple roles. You can see the user's roles on the user's profile.

# Role properties

Every role has these properties:

- `Owner`
- `Name`
- `CreatedTime`
- `DisplayName`
- `IsEnabled`
- `Users` Array of this role's sub users

- `Roles` Array of this role's sub roles

# Permissions

Each user may have multiple permissions. You can see the user's permissions on the user's profile.

# Permission properties

Premision has these properties:

- `Owner`
- `Name`
- `CreatedTime`
- `DisplayName`
- `IsEnabled`
- `Model`

- `Users` Array of this role's sub users
- `Roles` Array of this role's sub roles
- `ResourceType`
- `Resources` Array of the resources
- `Actions` Array of the actions
- `Effect`

# Syncer

## Overview

Synchronize users in Casdoor

## Database

Using Databse Syncer to synchronize database

## Keycloak

Using Keycloak Syncer to synchronize Keycloak

# Overview

As an authentication platform, Casdoor can easily manipulate users stored in databases.

## Syncer

Casdoor stores users in **user** table. Don't worry about migrating your application **user** data into Casdoor, when you plan to use Casdoor as an authentication platform. Casdoor provides **syncer** to quickly help you sync user data to Casdoor.

Specify the database and user table that you want to synchronize to Casdoor. And the syncer will sync the data after the specified interval. For details, see database syncer.

## Synchronization hash

Casdoor use hash to determine how to update a user. Casdoor would calculate the hash value of each user in the table, which is generated using users' information, such as password or mobile phone number.

If the calculated hash value of a user with a specific `Id` changed compared with the original value, Casdoor would affirm which user table has been updated. Then the database would update the old information, realize the **bilateral synchronization** between Casdoor user table and origin user table.

# Database

## Database Syncer

The users table we created as a demo are imported from the template XLSX file.

| owner | name | created_time | updated_time | id | type | password | password_salt | display_name | first_name | last_name | avatar | permanent_avata | email |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| built-in | einstein | 2022-03-20T20:46:29+08:00 | | 1c57cc37-37f5-4def-9e9f-082189ef63d2 | normal-user | 123 | | Albert Einstein | | | https://casbin.org | | z6mive@ |
| built-in | euler | 2022-03-20T20:47:08+08:00 | | bb7831b4-0d24-4e96-b043-f8fd8d15eb | normal-user | 123 | | Leonhard Euler | | | https://casbin.org | | 3dzw4j@ |
| built-in | galieleo | 2022-03-20T20:47:58+08:00 | | 7920eb6c-f9f5-40ef-8e18-3ac99f49bdf5 | normal-user | 123 | | Galileo Galilei | | | https://casbin.org | | 8p4f38@ |
| built-in | gauss | 2022-03-20T20:48:33+08:00 | | f0c288f6-2c0d-479b-b545-cb4cf96db36 | normal-user | 123 | | Carl Friedrich Ga | | | https://casbin.org | | vqdsan@ |
| built-in | tesla | 2022-03-20T20:49:03+08:00 | | 687c3068-fd21-4d32-b2ba-e13e8b369a | normal-user | 123 | | Nikola Tesla | | | https://casbin.org | | 9v73hn@ |

Click the **Syncers** tab and create a new syncer. Fill in all the required information as below and save.

> TIP
>
> In general, you need to fill in at least the `ID` and `Name` in Casdoor Columns. And others important fields like `createdTime`, `Password`, `DisplayName`.

The following are the required fields.

- `Organization`: The organization that the user will import
- `Name`: The syncer name
- `Type`: Select database
- `Host`: The original database host
- `Port`: The original database port
- `User`: The original database username
- `Password`: The original database password
- `Database type`: All Xorm supported databases, like: MySQL, PostgreSQL, SQL Server, Oracle, Sqlite
- `Database`: The original database name
- `Table`: The original user table name
- `Table columns`
- `Column name`: The original user column name
- `Column type`: The original user column type
- `Casdoor Column`: The casdoor user column name

Optional fields

- `Is hashed`: Whether to calculate hash value. When enable "**Is hashed**", if the field of user in origin table updated, the syncer will sync this user. Disable "Is hashed", meaning if only the field update, the syncer need not sync the user. In short, the user does not synchronize until the fields involved in the hash calculation(enable "Is hashed") are updated.

- `Avatar base URL`: When sync users, if **Avatar base URL** is not empty and origin **user.avatar** not hasPrefix "http", new **user.avatar** will be replaced by **Avatar base URL + user.avatar**.
- `Affiliation table`: It is used to sync the affiliation of user from this table in database. Because the affiliation may be code of int type in "Affiliation table", so we need to map the int to a string. See getAffiliationMap() . Because Casdoor has some redundant fields to borrow, here we use `score` to map the int code to a string name.

Then you can turn on the **Is enable** button and save, the syncer will start to work.

# Keycloak

## Keycloak Syncer

The Keycloak syncer is basically the same as the database syncer, except that the `Table` and `Table columns` can be configured automatically for Keycloak.

In addition, the Keycloak syncer will query `credential` table, `keycloak_group` table and `user_group_membership` table because the user information in Keycloak is stored in multiple tables.

# Tokens

## Overview

Introduction to tokens in Casdoor

# Overview

Casdoor is based on OAuth. Tokens are users' OAuth token.

- `Owner`
- `Name`
- `CreatedTime`
- `Application`
- `Organization`
- `User`
- `Code`
- `AccessToken`
- `ExpireIn` Tokens will expire in hours
- `Scope` Scope of authorization
- `TokenType` E.g. type `Bear`

There are two options to generate a JWT Token after logging into the application:

- `JWT`
- `JWT-Empty`

The JWT option will create a token with all `User` fields. The `JWT-Empty` will create a token with all non-empty values for the user.

# Webhooks

## Overview

Add webhooks in Casdoor

# Overview

## Overview

Event systems allow you to build integrations, which subscribe to certain events on Casdoor. When one of those event is triggered, we'll send a POST json payload to the configured URL. The application parsed the json payload and carry out the hooked function. Events consist of signup, login, logout, update users, which are stored in the action field of the record. Event systems can be used to update an external issue from users.

# Deploy

## Nginx

use Nginx to reverse proxy your backend Go program, quickly start the Casdoor service

## k8s

Deploy Casdoor in k8s

# Nginx

Although Casdoor is a front-end back-end separation architecture, in the production environment, the back-end program still provides static file services for front-end files. Therefore, you can use reverse proxy software such as Nginx to proxy all traffic for the Casdoor domain and redirect it to the port monitored by the backend go program.

In this chapter you will learn how to use Nginx to reverse proxy your backend Go program, and quickly start the Casdoor service.

## 1. Build front end static files

Now assume that you have downloaded Casdoor and completed the necessary configuration. If not, go to **Get started** section.

You only needs to build static files, like this:

Yarn npm

```
yarn install && yarn run build
```

```
npm install && npm run build
```

# 2. Run the back end program

```
go run main.go
```

Or build first:

```
go build && ./main
```

# 3. Configure and run Nginx

```
vim /path/to/nginx/nginx.conf
```

Add a server:

```
server {
    listen 80;
    server_name YOUR_DOMAIN_NAME;
    location / {
        proxy_set_header    Host            $http_host;
        proxy_set_header    X-Real-IP       $remote_addr;
        proxy_set_header    X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_redirect      off;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Then restart your nginx process, run:

```
nginx -s reload
```

# 4. Test

Visit `http://YOUR_DOMAIN_NAME` in your favorite browser.

# k8s

## Deploy Casdoor in k8s

We have given a basic example of deploying Casdoor into k8s. In the root folder of casdoor, there exists a file named "k8s.yaml", which includes an example minimum configuration to be used in deploying casdoor in k8s, a deployment and a service.

First, make sure that you have modified the conf/app.conf so that the casdoor can successfully connect to the database, and the database is running. Second, make sure k8s is able to pull the necessary images.

Run

```
kubectl apply -f k8s.yaml
```

And soon you can see the result via command `kubectl get pods`

The content of k8s.yaml is as follow

```yaml
# this is only an EXAMPLE of deploying casddor in kubernetes
# please modify this file according to your requirements
apiVersion: v1
kind: Service
metadata:
  #EDIT IT: if you don't want to run casdoor in default namespace,
please modify this field
  #namespace: casdoor
  name: casdoor-svc
```

This file is merely an example. For example, you can choose to use a namespace other than default, use a service type instead of nodeport to expose the casdoor, or use a use config map in k8s to mount the configuration file, which is a more recommended way in k8s.

# LDAP

## Overview

Casdoor cooperates with a ldap server

## Config and Sync LDAP Users

LDAP configuration in Casdoor

## LDAP Server

How to connect ldap client in Casdoor

# Overview

Support for ldap server currenty has been introduced into Casdoor. Casdoor is able to synchronize users from ldap servers to Casdoor to use them as user accounts to log in, and authenticate them using the ldap servers. Casdoor also supports setting up cron job to synchronize users automatically on a regular basis.

## Detail about Casdoor-Ldap synchroniztion mechanism

How Casdoor cooperates with a ldap server is described as follows:

1. Synchronization: Casdoor can connect to ldap server fetch users' information, reads all users' information (include `uidNumber`,`uid`, `cn`, `gidNumber`, `mail`, `email`, `emailAddress`, `telephoneNumber`, `mobile`, `mobileTelephoneNumber`, `registeredAddress`, `postalAddress`), and creates Casdoor accounts for each user in ldap, and stores the accounts in database.

2. Authentication: As we have seen, Casdoor doesn't fetch ldap users' passwords to Casdoor. Thus, when the account which is synchronized from ldap server tries to log in through Casdoor, instead of checking password stored in casdoor's database, Casdoor connects to ldap server and verifies whether the user's password is correct.

3. User distinguished: Casdoor uses `uid` to exclusively identify a user, thus please make sure every ldap user has a different uid.

Once a user is synchronized into Casdoor,that user's information is detached with the ldap user, which means, if you modify the user's information in Casdoor, the

user's information in ldap won't be modified and vice versa (except ldap user's password, we rely on it to authenticate a user)

# Config and Sync LDAP Users

Ldap configurations belong to an organization, which ldap users will be synchronized into.

You are supposed to use a global admin user to modify the configuration. You need to enter the following information of the LDAP user synchronization in the "organization" page.

# Config to connect LDAP server

### Server Name

A friendly name is used by managers to identify different servers.

> e.g: `Example LDAP Server`

### Server Host

LDAP server's host or IP address.

> e.g: `example.com`

### Server Port

LDAP server's ports, only allow numbers.

> e.g: `389`

### Base DN

Casdoor uses Sub search mode by default when searching in LDAP. Base DN is the basic distinguished name of the search. Casdoor will return all users under the current base DN.

The admin account configured in casdoor should have at least read-only permissions at base DN.

> e.g: `ou=Example,dc=example,dc=com`

### Search filter

Casdoor uses search filter to query ldap users.

> e.g: `(objectClass=posixAccount)`

### Filter fields

Filter fields are the identifier of the user in LDAP server, When you log in to Casdoor as an LDAP user. Casdoor regards the entered login username as the `uid` of LDAP user. You can also config other filed, such as `mail`, `mobile`.

### Admin

An account that can log in to the specified LDAP server.

Login with DN or ID depends on the LDAP server settings you want to connect.

> e.g: `cn=manager,dc=example,dc=com`

### Admin Password

Password of LDAP server Admin account.

### Auto Sync

Set `0` to disable auto sync, other value means **Sync every few minutes**.

# Sync users

The sync table displays all users get from the LDAP server in the specific `ou`. If

the users have been synced, the checkbox will be disabled. You can check the box to select users, then sync the selected users from the LDAP server.

> **CAUTION**
>
> If the `uid` of the user in LDAP server is same as the `name` of a user existed in the organization of Casdoor, Casdoor will create a new user that the `name` include the `uid` and a random string. But the user may can not to login, because the name of the new synced user not exist in LDAP server. So try to avoid that.

# LDAP Server

Many systems like `Nexus` support `ldap` authentication. A simple ldap server is also implemented in Casdoor, which supports bind and search operations.

The following describes how to connect to the ldap server in Casdoor and implement simple login authentication.

## LDAP Server Port

The LDAP server listens on port `389` by default, you can change the default port by changing `ldapServerPort` in conf/app.conf.

## How it works

Similar to the ldap client in Casdoor, the users in the ldap server are all subclasses of `poxisAccount`.

When the server receives a set of data transmitted by the ldap, it will parse the `cn` and `ou`, where `cn` represents the username, `ou` represents the organization name. It doesn't matter what `dc` is.

If it is a bind operation, the server will use Casdoor to verify the username and password and give the user permission in Casdoor.

If it is a search operation, the server will check whether the Search operation is legal according to the permissions given to the client by the bind operation and return a response.

> INFO
>
> We only support **`Simple Authentication`**.

## How to bind

In Casdoor ldapserver, we only recognize `DN` similar to this: `cn=admin,ou=built-in,dc=example,dc=com`.

So please set the `DN` of the admin user to the above form. Then you can use this `DN` to bind to ldap server with the user's password to log in to casdoor for verification. If the server verification is passed, the user will be granted the authority in Casdoor.

## How to search

Once the bind operation completes successfully, you can perform the correct search operation. There are some differences between search and bind.

- If you want to search for a certain user, such as `Alice` under the `built-in` organization, you should use `DN` like this :`ou=built-in,dc=example,dc=com`, and add `cn=Alice` in the Filter field.
- If you want to search for all users under a certain organization, such as all users in `built-in`, you should use `DN` like this :`ou=built-in,dc=example,dc=com`, and add `cn=*` in the Filter field.
- If you want to search for all users for all organizations (the premise is that the user has sufficient permissions), you should use `DN` like this :`ou=*,dc=example,dc=com`, and add `cn=*` in the Filter field.

# Integrations

## Go

8 items

## Java

16 items

## JavaScript

1 items

## Lua

1 items

## 🗂️ PHP

4 items

## 🗂️ Ruby

1 items

## 🗂️ Haskell

1 items

## 🗂️ Python

1 items

# Go

## Kubernetes

Using Casdoor for authentication in Kubernetes

## OpenShift

Using Casdoor for authentication in openshift

## BookStack

Using Casdoor for authentication in BookStack

## Bytebase

Using OAuth2 to connect various applications, like Bytebase

## ELK

Overview of casdoor/elk-auth-casdoor

## Gitea

Using Casdoor for authentication in Gitea

## Grafana

Using Casdoor for authentication in Grafana

## MinIO

Configuring Casdoor as identity provider to support with MinIO

# Kubernetes

According to the Kubernetes documentation, the API Server of Kubernetes can be authenticated using OpenID Connect (OIDC). This article will guide you on how to configure authentication in Kubernetes using Casdoor.

## Environment Requirements

Before starting, please make sure that you have the following environment:

- A Kubernetes cluster.
- A Casdoor application like this demo website
- kubectl command tool (optional)

> NOTE
>
> Kubernetes oidc-issuer-url only accepts URLs which use the https:// prefix. So your Casdoor application should be deployed on an HTTPS website.

## Step 1: Creating an Casdoor App and User Account for Authentication

Go to your Casdoor and add your new application **Kubernetes**. Please remember the `Name`, `Organization`, `client ID`, `client Secret` and add some grant type in this APP.

Next, we will add a new user to the application that we just created. Please note that the `organization` and `Signup application` used here should correspond to the APP registered earlier.

Organization ⓘ : casbin

ID ⓘ : 202e02e9-9128-496a-a209-fdb336448f56

Name ⓘ : user_pnvm5i

Display name ⓘ : New User - pnvm5i

Avatar ⓘ :

Preview:

Upload a photo...

User type ⓘ : normal-user

Password ⓘ : Modify password...

Email ⓘ : pnvm5i@example.com

Phone ⓘ : +1 78005961394

Country/Region ⓘ : Please select country/region

Location ⓘ :

Affiliation ⓘ : Example Inc.

Title ⓘ :

Homepage ⓘ :

Bio ⓘ :

Tag ⓘ : staff

Signup application ⓘ : Kubernetes

# Step 2: Configure Kubernetes API Server with OIDC Authentication

To enable the OIDC plugin, at least configure the following flags on the API server:

- --oidc-issuer-url : URL of the provider which allows the API server to discover public signing keys.
- --oidc-client-id : A client id that all tokens must be issued for.

This article use of minikube for demonstration. We can configure the OIDC plugin for the minikube's API server using the following command at startup:

```
minikube start --extra-config=apiserver.oidc-issuer-
url=https://demo.casdoor.com  --extra-config=apiserver.oidc-client-
id=294b09fbc17f95daf2fe
```

# Step 3: Test OIDC Authentication

## Obtain Authentication Information

Due to the lack of a frontend in kubectl, authentication can be performed by sending a POST request to the Casdoor server. Here is the code in Python which sends a POST request to the Casdoor server and retrieves the `id_token` and `refresh_token`:

```
import requests
import json
```

After executing this code, you should receive a response similar to the following:

```
{
  "access_token": "xxx",
  "id_token": "yyy",
  "refresh_token": "zzz",
  "token_type": "Bearer",
  "expires_in": 72000,
  "scope": ""
}
```

Now, we can use the `id_token` that we just obtained to authenticate with Kubernetes API server.

## HTTP Request-Based Authentication

Add the token to the request header.

```
curl https://www.xxx.com -k -H "Authorization: Bearer $(id_token)"
```

- https://www.xxx.com is the Kubernetes API server deployment address.

## Kubectl Client-Based Authentication

### Configuration File Method

Write the following configuration to the ~/.kube/config file. You should replace each configuration item in the configuration file above with the values you obtained earlier.

```
users:
```

Now you can directly access your API server using kubectl. Try running a test command.

```
kubectl cluster-info
```

### Command Line Argument Method

Alternatively, you can authenticate by directly adding the id_token to the command line parameters of kubectl.

```
kubectl --token=$(id_token) cluster-info
```

# OpenShift

OpenShift supports OIDC, so we can integrate Casdoor with OpenShift. The following steps demonstrate how to integrate Casdoor with OpenShift Local using the online demo of Casdoor.

## Step1. Create an Casdoor application

Add a new application in Casdoor, note following points.

- Remember the `Client ID` and `Client secret` for the next step.
- The format of the Redirect URL is `https://oauth-openshift.apps.<cluster_name>.<cluster_domain/*`, Fill it in depending your situation

# Step2. Openshift Oauth Configuration

Now, login into the Openshift Console as Kubeadmin. Once you are logged In. Browse to the side menu, locate the Cluster settings

Under Global Configuration You will see Oauth

You will see the Identity Provider section. In ADD section, select the OpenID Connect from options.

Configure OIDC, note following points.

- Fill in the `Client ID` and `Client Secret` remembered in the previous step.
- The Issuer URL must use https, with the form `https://<casdoor-host>`, again depending on your situation

## Add Identity Provider: OpenID Connect

Integrate with an OpenID Connect identity provider using an Authorization Code Flow.

**Name** *

casdoor

Unique name of the new identity provider. This cannot be changed later.

**Client ID** *

2452f2b5abb6ff131199

**Client secret** *

••••••••••••••••••••••••••••••••••••••••••••

**Issuer URL** *

https://demo.casdoor.com

The URL that the OpenID provider asserts as its issuer identifier. It must use the https scheme with no URL query parameters or fragment.

# Step3. Test OIDC Authentication

Access the Openshift console in the browser. You will see casdoor (The Name you configured in the previous step). Click on the casdoor login option. You will get redirected to the Casdoor login page.

# BookStack

## Using Casdoor for authentication in BookStack

BookStack is an open source book and document sharing site, as well as an open source application developed using the Go language to help you better achieve document reading management.

Bookstack-casdoor has been integrated with **Casdoor**, and you can now start quickly with a simple configuration.

### Step1. Create an Casdoor application

Go to your Casdoor and add your new application **BookStack**. Here is an **example** of creating the BookStack application in Casdoor.

Please remember the `Name`, `Organization`, `client ID`, and `client Secret`. You will use them in the next step.

## Step2. Configure Casdoor Login

Now, please move to the BookStack. Find the file: `oauth.conf.example`.

Rename `oauth.conf.example` to `oauth.conf` and **modify** the configuration. The content of which by default is:

```
[oauth]

casdoorOrganization =  <"Organization">
casdoorApplication = "bookstack"
casdoorEndpoint = http://localhost:8000
clientId =   <client ID>
clientSecret =  <client Secret>
redirectUrl = http://localhost:8181/login/callback
```

## Step3. Fill in the `redirectUrl` in Casdoor

The last step, go back to the page where you added the **BookStack application**, and fill in the `Redirect URLs`. Make sure the `Redirect URL` is the same as the `redirectUrl` in the file `oauth.conf`.

**Now that you've done all the configuration for Casdoor!**

You can go back to your BookStack and experience using Casdoor for login authentication once the BookStack has been successfully deployed.

# Bytebase

Casdoor can use OAuth2 to connect various applications. Here we will use Bytebase as an example to show you how to use OAuth2 to connect to your applications.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Bytebase_HOSTNAME`: Domain name or IP where Bytebase is deployed.

## Step1. Deploy Casdoor and Bytebase

Firstly, the Casdoor and Bytebase should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find a redirect url: `<CASDOOR_HOSTNAME>/oauth/callback`
3. Add your redirect url to casdoor application:

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Bytebase

1. You should find sso and use OAuth 2.0

2. You should config this app

3. You can find Client Id and Client Secret in Casdoor application page.

- `Token server url`: http://**`CASDOOR_HOSTNAME`**/api/login/oauth/access_token
- `Authorization server url`: http://**`CASDOOR_HOSTNAME`**/login/oauth/authorize
- `UserInfo server url`: http://**`CASDOOR_HOSTNAME`**/api/get-account
- `Scopes`: `address phone openid profile offline_access email`

Log out of Bytebase, and test SSO.

# ELK

## Overview of casdoor/elk-auth-casdoor

One of the biggest drawbacks of ELK (Elasticsearch, Logstash and Kibana) is that originally, these products have no authentication mechanism, so that everyone can visit the kibana dashboard as long as he has the url of kibana, or ES urls. Later ELK integrated an embedded authentication system "Xpack", whose all advanced functions **are not free** (eg. Oauth, OIDC, LDAP, SAML), and only plain authentication (setting a set of accounts and passwords) is free of charge, which is quite inconvenient. We cannot just provide a unique account for everyone in a corporation.

Therefore, we have developed a elk authentication solution based on Casdoor, **free of charge, open source and under maintenance, supporting lots of advanced features**. Casdoor is a centralized authentication/ Single-Sign-On platform based on Oauth2.0/OIDC, and casdoor/elk-auth-casdoor is actually a reverse proxy, which is designed to intercept all http data flow toward the elk/kibana, and guides the users who haven't logged in to log in. This reverse proxy is completely transparent as long as the user has logged in.

If this user hasn't been correctly authenticated, the request will be temporally cached, and the user will be redirected to Casdoor login page. After user logs in through casdoor, the cached request will be restored and sent to kibana. So it's ok if a POST request (or something other than GET) is intercepted, and user won't need to refill the form and resend the request. The reverse proxy will remember it for you.

Location of casdoor/elk-auth-casdoor repository https://github.com/casdoor/elk-

auth-casdoor

# How to run it?

0. have golang environment installed
1. go to casdoor/elk-auth-casdoor and fetch the code
2. register your proxy as an app of Casdoor.
3. modify the configuration

The configuration file locates in "conf/app.conf". Here is an example, and you should customize changes according to your real demands.

```
appname = .
# port on which the reverse proxy shall be run
httpport = 8080
runmode = dev
#EDIT IT IF NECESSARY. The url of this reverse proxy
pluginEndpoint="http://localhost:8080"
#EDIT IT IF NECESSARY. The url of the kibana
targetEndpoint="http://localhost:5601"
#EDIT IT. The url of casdoor
casdoorEndpoint="http://localhost:8000"
#EDIT IT. The clientID of your reverse proxy in casdoor
clientID=ceb6eb261ab20174548d
#EDIT IT. The clientSecret of your reverse proxy in casdoor
clientSecret=af928f0ef1abc1b1195ca58e0e609e9001e134f4
#EDIT IT. The application name of your reverse proxy in casdoor
appName=ELKProxy
#EDIT IT. The organization to which your reverse proxy belongs in
casdoor
organization=built-in
```

4. visit http://localhost:8080 (in the example above), and log in following the guidance of redirection, and you shall see kibana protected and authenticated by casdoor.

5. If everything works well, don't forget to block the visits of original kibana's port coming from outside by configurating your firewall(or something else), so that outsiders can only visit kibana via this reverse proxy.

# Gitea

## Using Casdoor for authentication in Gitea

Gitea is a community managed lightweight code hosting solution written in Go. It is published under the MIT license.

Gitea supports 3rd-party authentication including Oauth, which makes it possible to use Casdoor to authenticate it. Here is the tutorial for achieving this.

### Preparations

To configure Gitea to use Casdoor as identification provider, you need to have Gitea installed as well as access to administrator account.

For more information about how to download, install and run Gitea see https://docs.gitea.io/en-us/install-from-binary/

You are supposed to create an administrator account during installation. If you didn't, the administrator will be the first registered user. Please use this account proceed the following procedures.

### 1. Create an Casdoor application

Like this

Please remember the client ID and client Secret for the next step.

Please don't fill in the callback url in this step. The url depends on the configurations on gitea in the next step. Later we will come back to set a correct callback url.

## 2. Configure Gitea to use Casdoor

Log in as administrator. Go to 'Site Administration' page via drop-down menu in the upper right corner. Then Switch to "Authentication Source" Page.

You are supposed to see something like this.

Press the "Add Authentication Source" Button, and fill in the form like this.

Please choose the authentication type as "oauth2".

Please input a name for this authentication source and **remember this name**. This name will be used for the callback_url in the next step.

Please choose the `OpenID Connect` Oauth2 Provider.

Fill in the **Client ID** and **Client Secret** remembered in the previous step.

Fill in the openid connect auto discovery url, which is supposed to be `<your endpoint of casdoor>/.well-known/openid-configuration`.

Fill in the other optional configuration items as you wish. And then submit it.

Submit the form.

## 3. Configure the callback url in casdoor

Go back to the application edit page in step 2, and add the following callback url:

`<endpoint of gitea>/user/oauth2/<authentication source name>/callback`

The `<authentication source name>`is the name for authentication source in Gitea in the previous step.

## 4. Have a try on Gitea

Logout the current administrator account.

You are supposed to see this in login page:

Press the 'sign in with openid' button and you will be redirected to casdoor login page.

After login you will see this:

Follow the instructions and bind the casdoor account with a new gitea account or

existing account.

Then everything will be working correctly.

# Grafana

## Using Casdoor for authentication in Grafana

Grafana supports authentication via Oauth. Therefore it is extremely easy for users to use casdoor to log in in Grafana. Only several steps and simple configurations can achieve that.

Here is a tutorial to use Casdoor for authentication in Grafana. Before you proceed, please ensure that you have grafana installed and running.

## Step 1 Create an app for Grafana in Casdoor

Here is an example of creating an app in Casdoor

Please copy the client secret and client id for the next step.

Please add the callback url of Grafana. By default, Grafana's oauth callback is `/login/generic_oauth`. So please concatenate this url correctly.

# Step 2: Modify the configuration of Grafana

By default the configuration file for oauth locates at `conf/defaults.ini` in the workdir of Grafana.

Please find the section `auth.generic_oauth` and modify the following field:

```
[auth.generic_oauth]
name = Casdoor
```

## About HTTPS

If you don't want HTTPS enabled for casdoor or if you deploy grafana without HTTPS enabled, please also set `tls_skip_verify_insecure = true`

## About redirectURI after Sign In With Casdoor

If the redirect uri is not right after Sign In with Casdoor in Grafana, you may want to configure root_url

```
[server]
http_port = 3000
# The public facing domain name used to access grafana from a
browser
domain = <your ip here>
# The full public facing url
root_url = %(protocol)s://%(domain)s:%(http_port)s/
```

related links:

1. Grafana doc
2. Grafana defaults.ini

## About Role Mappping:

You may want to configure role_attribute_path to map your user's role to Grafana via role_attribute_path

```
[auth.generic_oauth]
role_attribute_path = contains(roles[*].name, 'admin') && 'Admin'
```

the JMESPath expression after role_attribute_path is very important here. read grafana doc please

# Step3: See whether it works

Shutdown grafana and restart it.

Go to see the login page, you are supposed to see something like this

# MinIO

MinIO supports external identity management using an OpenID Connect (OIDC) -compatible provider. This document covers configuring Casdoor as identity provider to support with MinIO.

## Step1. Deploy Casdoor & MinIO

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

You can refer to here to deploy your MinIO server and here for MinIO client called `mc`.

## Step2. Configure Casdoor Application

1. Create or use an existing Casdoor application.

2. Add Your redirect url

3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

4. This step is necessary for MinIO. As MinIO needs to use a claim attribute in JWT for its policy, you should configure it in casdoor as well. Currently, casdoor uses `tag` as a workaround for configuring MinIO's policy.

You can find all supported policies here.

# Step3. Configure MinIO

You can start a MinIO server by following commands:

```
export MINIO_ROOT_USER=minio
export MINIO_ROOT_PASSWORD=minio123
minio server /mnt/export
```

You can use parameter `--console-address` to configure the address and port.

Then you can add a service alias by MinIO client `mc`.

```
mc alias set myminio <You console address> minio minio123
```

Now, you can configure OpenID connect of MinIO. For Casdoor, the command is like the following:

```
mc admin config set myminio identity_openid
config_url="http://CASDOOR_HOSTNAME/.well-known/openid-
configuration" client_id=<client id> client_secret=<client secret>
claim_name="tag"
```

You can refer to offical document for more detailed parameters.

Once successfully set, restart the MinIO instance.

```
mc admin service restart myminio
```

# Step4. Try the demo!

Now, you can open your MinIO console in the browser and click on `Login with SSO`.

You will be redirected to the casdoor user login page. Upon successful login you

will be redirected to MinIO page and logged in automatically and you should see now the buckets and objects they have access to.

> ⚠ CAUTION
>
> If you deploy frontend and backend of casdoor in different ports, the login page you are redirected to will be backend port and it will display `404 not found`. You can modify the port to the frontend one. Then you can access to casdoor login page successfully.

# Java

## Spring Boot

Using Casdoor in Spring Boot project

## Spring Cloud

Using Casdoor in Spring Cloud

## Spring Cloud Gateway

Using Casdoor in Spring Cloud Gateway

## Spring Security

2 items

## Jenkins Plugin

Using Casdoor plugin for your Jenkins security

## Jenkins OIDC

Using OIDC protocol as IDP to connect various applications, like Jenkins

## Jira

2 items

## Confluence

Using OIDC protocol as IDP to connect various applications, like Confluence

## RuoYi

Using Casdoor in RuoYi-Cloud

## Pulsar-manager

Using Casdoor in Pulsar-manager

## ShenYu

Using Casdoor in ShenYu

## ShardingSphere

Using Casdoor in ShardingSphere

## Apache IoTDB

Using Casdoor Apache IoTDB

## Apache DolphinScheduler

Using Casdoor for DolphinScheduler SSO login

## FireZone

Using OIDC protocol as IDP to connect various applications, like FireZone

## CloudFoundry

Learn how to integrate Casdoor with CloudFoundry to secure your applications

# Spring Boot

casdoor-spring-boot-example is an example of how to use casdoor-spring-boot-starter in SpringBoot project. We will show you the steps below.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation. Please deploy your Casdoor instance in **production mode**.

After a successful deployment, you need to ensure:

- Open your favorite browser and visit **http://localhost:8000**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor-based login page in your own app with the following steps.

## Step2. Import casdoor-spring-boot-starter

You can import the casdoor-spring-boot-starter with maven or gradle.

Maven Gradle

```
<!-- https://mvnrepository.com/artifact/org.casbin/casdoor-spring-boot-starter -->
<dependency>
    <groupId>org.casbin</groupId>
    <artifactId>casdoor-spring-boot-starter</artifactId>
    <version>1.x.y</version>
</dependency>
```

```
// https://mvnrepository.com/artifact/org.casbin/casdoor-spring-boot-starter
implementation group: 'org.casbin', name: 'casdoor-spring-boot-starter', version: '1.x.y'
```

# Step3. Init Config

Initialization requires 6 parameters, which are all string type.

| Name (in order) | Must | Description |
|---|---|---|
| endpoint | Yes | Casdoor Server Url, such as `http://localhost:8000` |
| clientId | Yes | Application.client_id |
| clientSecret | Yes | Application.client_secret |
| certificate | Yes | Application.certificate |

| Name (in order) | Must | Description |
| --- | --- | --- |
| organizationName | Yes | Application.organization |
| applicationName | No | Application.name |

You can use Java properties or YAML files to init as below.

Properties YML

```
casdoor.endpoint = http://localhost:8000
casdoor.clientId = <client-id>
casdoor.clientSecret = <client-secret>
casdoor.certificate = <certificate>
casdoor.organizationName = built-in
casdoor.applicationName = app-built-in
```

```
casdoor:
  endpoint: http://localhost:8000
  client-id: <client-id>
  client-secret: <client-secret>
  certificate: <certificate>
  organization-name: built-in
  application-name: app-built-in
```

> ⚠ CAUTION
>
> You should replace the configuration with your own Casdoor instance, especially the `clientId`, `clientSecret` and the `jwtPublicKey`.

# Step4. Redirect to the login page

When you need the authentication who access your app, you can send the target url and redirect to the login page provided by Casdoor.

Please be sure that you have added the callback url (e.g. **http://localhost:8080/login**) in application configuration in advance.

```
@Resource
private CasdoorAuthService casdoorAuthService;

@RequestMapping("toLogin")
public String toLogin() {
    return "redirect:" +
casdoorAuthService.getSigninUrl("http://localhost:8080/login");
}
```

# Step5. Get token and parse

After Casdoor verification passed, it will be redirected to your application with code and state.

You can get the code and call `getOAuthToken` method, then parse out jwt token.

`CasdoorUser` contains the basic information about the user provided by Casdoor. You can use it as a keyword to set the session in your application.

```
@RequestMapping("login")
public String login(String code, String state, HttpServletRequest
request) {
```

# Service

Examples of APIs are shown below.

- CasdoorAuthService
  - `String token = casdoorAuthService.getOAuthToken(code, "app-built-in");`
  - `CasdoorUser casdoorUser = casdoorAuthService.parseJwtToken(token);`
- CasdoorUserService
  - `CasdoorUser casdoorUser = casdoorUserService.getUser("admin");`
  - `CasdoorUser casdoorUser = casdoorUserService.getUserByEmail("admin@example.com");`
  - `CasdoorUser[] casdoorUsers = casdoorUserService.getUsers();`
  - `CasdoorUser[] casdoorUsers = casdoorUserService.getSortedUsers("created_time", 5);`
  - `int count = casdoorUserService.getUserCount("0");`
  - `CasdoorResponse response = casdoorUserService.addUser(user);`
  - `CasdoorResponse response = casdoorUserService.updateUser(user);`
  - `CasdoorResponse response = casdoorUserService.deleteUser(user);`
- CasdoorEmailService
  - `CasdoorResponse response = casdoorEmailService.sendEmail(title, content, sender, receiver);`
- CasdoorSmsService
  - `CasdoorResponse response = casdoorSmsService.sendSms(randomCode(), receiver);`

- CasdoorResourceService
  - `CasdoorResponse response = casdoorResourceService.uploadResource(user, tag, parent, fullFilePath, file);`
  - `CasdoorResponse response = casdoorResourceService.deleteResource(file.getName());`

# What's more

You can explore the following projects/docs to learn more about the integration of Java with Casdoor.

- casdoor-java-sdk
- casdoor-spring-boot-starter
- casdoor-spring-boot-example
- casdoor-spring-security-example
- casdoor-spring-security-react-example
- casdoor-spring-boot-shiro-example

# Spring Cloud

Under the Spring Cloud microservice system, general authentication occurs at the gateway. Please refer to casdoor-springcloud-gateway-example.

If you want to use Casdoor in a single service, you can refer to casdoor-spring-boot-example.

No matter in the gateway layer or in the single service, both use casdoor-spring-boot-starter.

## What's more

You can explore the following projects/docs to learn more about the integration of Java with Casdoor.

- casdoor-java-sdk
- casdoor-spring-boot-starter
- casdoor-spring-boot-example
- casdoor-spring-security-example
- casdoor-spring-security-react-example
- casdoor-spring-boot-shiro-example
- casdoor-springcloud-gateway-example

# Spring Cloud Gateway

casdoor-springcloud-gateway-example is an example on how to use casdoor-spring-boot-starter as a OAuth2 plugin in Spring Cloud Gateway. We will show you the steps below.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation. Please deploy your Casdoor instance in **production mode**.

After a successful deployment, you need to ensure:

- Open your favorite browser and visit **http://localhost:8000**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2: Init a Spring Cloud Gateway

You can use the code of this example directly or combine your own business code.

We need a gateway service and at least one business service.

In this example, `casdoor-gateway` as the gateway service and `casdoor-api` as the business service.

# Step3: Include the dependency

Add `casdoor-spring-boot-starter` to the Spring Cloud Gateway project.

For Apache Maven:

/casdoor-gateway/pom.xml

```
<!-- https://mvnrepository.com/artifact/org.casbin/casdoor-spring-boot-starter -->
<dependency>
    <groupId>org.casbin</groupId>
    <artifactId>casdoor-spring-boot-starter</artifactId>
    <version>1.x.y</version>
</dependency>
```

For Gradle:

```
// https://mvnrepository.com/artifact/org.casbin/casdoor-spring-boot-starter
implementation group: 'org.casbin', name: 'casdoor-spring-boot-starter', version: '1.x.y'
```

# Step4: Configure your properties

Initialization requires 6 parameters, which are all string type.

| Name (in order) | Must | Description |
|---|---|---|
| endpoint | Yes | Casdoor Server Url, such as `http://localhost:8000` |
| clientId | Yes | Application.client_id |
| clientSecret | Yes | Application.client_secret |
| certificate | Yes | Application.certificate |
| organizationName | Yes | Application.organization |
| applicationName | No | Application.name |

You can use Java properties or YAML files to init as below.

For properties:

```
casdoor.endpoint=http://localhost:8000
casdoor.clientId=<client-id>
casdoor.clientSecret=<client-secret>
casdoor.certificate=<certificate>
casdoor.organizationName=built-in
casdoor.applicationName=app-built-in
```

For yaml:

```
casdoor:
  endpoint: http://localhost:8000
  client-id: <client-id>
```

In addition, you need to configure Gateway Routing. For yaml:

```yaml
spring:
  application:
    name: casdoor-gateway
  cloud:
    gateway:
      routes:
        - id: api-route
          uri: http://localhost:9091
          predicates:
            - Path=/api/**
```

# Step5: Add the CasdoorAuthFilter

Add an implementation class of GlobalFilter to the gateway for identity verification, such as CasdoorAuthFilter in this example.

If the authentication fails, it returns to the front end 401 to jump to the login interface.

```java
@Component
public class CasdoorAuthFilter implements GlobalFilter, Ordered {

    private static final Logger LOGGER =
LoggerFactory.getLogger(CasdoorAuthFilter.class);

    @Override public int getOrder() {
        return 0;
    }

    @Override public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {
```

# Step6: Get the Service and use

Now provide 5 services: `CasdoorAuthService`, `CasdoorUserService`, `CasdoorEmailService`, `CasdoorSmsService` and `CasdoorResourceService`.

You can create them as below in Gateway project.

```
@Resource
private CasdoorAuthService casdoorAuthService;
```

When you need the authentication who access your app, you can send the target url and redirect to the login page provided by Casdoor.

Please be sure that you have added the callback url (e.g. http://localhost:9090/callback) in application configuration in advance.

```
@RequestMapping("login")
public Mono<String> login() {
    return Mono.just("redirect:" +
casdoorAuthService.getSigninUrl("http://localhost:9090/callback"));
}
```

After Casdoor verification passed, it will be redirected to your application with code and state.

You can get the code and call `getOAuthToken` method, then parse out jwt token.

`CasdoorUser` contains the basic information about the user provided by Casdoor, you can use it as a keyword to set the session in your application.

```java
@RequestMapping("callback")
public Mono<String> callback(String code,String
state,ServerWebExchange exchange) {
    String token = "";
    CasdoorUser user = null;
    try {
        token = casdoorAuthService.getOAuthToken(code,state);
        user = casdoorAuthService.parseJwtToken(token);
    } catch(CasdoorAuthException e) {
        e.printStackTrace();
    }
    CasdoorUser finalUser = user;
    return exchange.getSession().flatMap(session -> {
        session.getAttributes().put("casdoorUser", finalUser);
        return Mono.just("redirect:/");
    });
}
```

Examples of APIs are shown below.

- CasdoorAuthService
  - `String token = casdoorAuthService.getOAuthToken(code, "app-built-in");`
  - `CasdoorUser casdoorUser = casdoorAuthService.parseJwtToken(token);`
- CasdoorUserService
  - `CasdoorUser casdoorUser = casdoorUserService.getUser("admin");`
  - `CasdoorUser casdoorUser = casdoorUserService.getUserByEmail("admin@example.com");`
  - `CasdoorUser[] casdoorUsers = casdoorUserService.getUsers();`
  - `CasdoorUser[] casdoorUsers = casdoorUserService.getSortedUsers("created_time", 5);`
  - `int count = casdoorUserService.getUserCount("0");`

    - `CasdoorResponse response = casdoorUserService.addUser(user);`
    - `CasdoorResponse response = casdoorUserService.updateUser(user);`
    - `CasdoorResponse response = casdoorUserService.deleteUser(user);`
- CasdoorEmailService
    - `CasdoorResponse response = casdoorEmailService.sendEmail(title, content, sender, receiver);`
- CasdoorSmsService
    - `CasdoorResponse response = casdoorSmsService.sendSms(randomCode(), receiver);`
- CasdoorResourceService
    - `CasdoorResponse response = casdoorResourceService.uploadResource(user, tag, parent, fullFilePath, file);`
    - `CasdoorResponse response = casdoorResourceService.deleteResource(file.getName());`

# Step7: Restart project

After start, open your favorite browser and visit **http://localhost:9090**, then click any button which can request resources from `casdoor-api`.

The gateway authentication logic will be triggered. Since you are not logged in, you will jump to the login interface. Click Login Button.

You can see the unified login platform of Casdoor.

After successful login, it will jump to the main interface. Then you can click any button.

## What's more

You can explore the following projects/docs to learn more about the integration of Java with Casdoor.

- casdoor-java-sdk
- casdoor-spring-boot-starter
- casdoor-spring-boot-example
- casdoor-spring-security-example
- casdoor-spring-security-react-example
- casdoor-spring-boot-shiro-example
- casdoor-springcloud-gateway-example

# Spring Security

## Spring Security OAuth

Using Spring Security as an example to show how to use OIDC to connect to your applications

## Spring Security Filter

Based on Spring Security Filter, how to use OIDC to connect your application.

# Spring Security OAuth

Casdoor can use OIDC protocol as IDP to connect various applications. Here we will use Spring Security as an example to show you how to use OIDC to connect to your applications.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add Your redirect url (You can see more details about how to get redirect url in the next section)

3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above. We will use them in next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Spring Security

Spring Security natively support OIDC.

You can customize the settings of Spring Security OAuth2 Client:

> ⚠ CAUTION
>
> You should replace the configuration with your own Casdoor instance, especially the `<Client ID>` and others.

application.yml  application.properties

```yaml
spring:
```

```
spring.security.oauth2.client.registration.casdoor.client-id=<Client
ID>
spring.security.oauth2.client.registration.casdoor.client-
secret=<Client Secret>
spring.security.oauth2.client.registration.casdoor.scope=<Scope>
spring.security.oauth2.client.registration.casdoor.authorization-
grant-type=authorization_code
spring.security.oauth2.client.registration.casdoor.redirect-
uri=<Redirect URL>

spring.security.oauth2.client.provider.casdoor.authorization-
uri=http://CASDOOR_HOSTNAME:7001/login/oauth/authorize
spring.security.oauth2.client.provider.casdoor.token-
uri=http://CASDOOR_HOSTNAME:8000/api/login/oauth/access_token
spring.security.oauth2.client.provider.casdoor.user-info-
uri=http://CASDOOR_HOSTNAME:8000/api/get-account
spring.security.oauth2.client.provider.casdoor.user-name-
attribute=name
```

> ⚠ CAUTION
>
> For default situation of Spring Security, the `<Redirect URL>` should be like `http://<You Spirng Boot Application Endpoint>/<Servlet Prefix if it is configured>/login/oauth2/code/custom`. For example, in the following demo, the redirect URL should be `http://localhost:8080/login/oauth2/code/custom`.
>
> You should also configure this in `casdoor` application.

You can also customize the settings by `ClientRegistration` in your code. You can find the mapping here

# Step4. Get Started with A Demo

1. We can create a Spring Boot application.

2. We can add a configuration which protects all endpoints except `/` and `/login**` for users to log in.

```
@EnableWebSecurity
public class UiSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers("/",
"/login**").permitAll().anyRequest().authenticated().and()
                .oauth2Login();

    }
}
```

3. We can add a naive page for user to log in.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Spring OAuth Client Thymeleaf - 1</title>
<link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/
bootstrap.min.css" />
</head>
<body>
    <nav
        class="navbar navbar-expand-lg navbar-light bg-light shadow-
sm p-3 mb-5">
```

When user clicks the `login` button, he will be redirected to `casdoor`.

4. Next, we can define our protected resources. We can export an endpoint called `/foos` and a web page for display.

Data Model

```
public class FooModel {
    private Long id;
    private String name;

    public FooModel(Long id, String name) {
        super();
        this.id = id;
        this.name = name;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Controller

```
@Controller
public class FooClientController {
    @GetMapping("/foos")
    public String getFoos(Model model) {
```

Web page

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Spring OAuth Client Thymeleaf - 1</title>
<link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/
bootstrap.min.css" />
</head>
<body>
    <nav
        class="navbar navbar-expand-lg navbar-light bg-light shadow-
sm p-3 mb-5">
        <a class="navbar-brand" th:href="@{/foos/}">Spring OAuth
Client
            Thymeleaf -1</a>
        <ul class="navbar-nav ml-auto">
            <li class="navbar-text">Hi, <span
sec:authentication="name">preferred_username</span>   
            </li>
        </ul>
    </nav>
    <div class="container">
        <h1>All Foos:</h1>
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <td>ID</td>
                    <td>Name</td>
                </tr>
            </thead>
            <tbody>
                <tr th:if="${foos.empty}">
                    <td colspan="4">No foos</td>
                </tr>
                <tr th:each="foo : ${foos}">
```

> ⚠ CAUTION
>
> All the web page template should be put under `resources/templates`.

# Step5. Try the demo!

Firstly, you can try to open your favorite browser and directly visit `/foos`. It will automatically redirect to casdoor's login page. You can log in here or from the root page.

If you visit your root page,

Click the `login` button and the page will redirect to casdoor's login page.

After you log in, the page will redirect to `/foos`.

# Spring Security Filter

Casdoor can use OIDC protocol as IDP to connect various applications. Here, we will use the filter in spring security to integrate casdoor and show you how to connect to the application using oidc.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add Your redirect url (You can see more detials about how to get redirect url in the next section).

3. On the certificate editing page, you can see your `Certificate`.

4. Add provider you want and supplement other settings.

Not surprisingly, you can get these values on the application settings page: `Application Name`, `Organization Name`, `Redirect URL`, `Client ID`, `Client Secret`, `Certification`. As shown above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Spring Security

You can customize the settings of spring security filters to process tokens:

> ⚠ CAUTION
>
> You should replace the configuration with your own Casdoor instance especially the `<Client ID>` and others.

```
server:
  port: 8080
```

> ⚠ CAUTION
>
> For frontend applications, the default value of `<FRONTEND_HOSTNAME>` is `localhost:3000`.
>
> For example, in the following demo, the redirect URL should be `http://localhost:3000/callback`.
>
> You should also configure this in `casdoor` application.

# Step4. Configure Frontend

You need to install `casdoor-js-sdk` and configure `SDK`.

1. Install `casdoor-js-sdk`.

```
npm i casdoor-js-sdk
# or
yarn add casdoor-js-sdk
```

2. Set up `SDK`.

```
import Sdk from "casdoor-js-sdk";

// Serverurl is the URL where spring security is deployed
export const ServerUrl = "http://BACKEND_HOSTNAME:8080";

const sdkConfig = {
  serverUrl: "http://CASDOOR_HOSTNAME:8000",
  clientId: "<your client id>",
  appName: "<your application name>",
  organizationName: "<your organization name>",
  redirectPath: "/callback",
```

# Step5. Get Started with A Demo

1. We can create a Spring Boot application.

2. We can add some configurations to handle JWT.

```
@EnableWebSecurity
public class SecurityConfig {

    private final JwtTokenFilter jwtTokenFilter;

    public SecurityConfig(JwtTokenFilter jwtTokenFilter) {
        this.jwtTokenFilter = jwtTokenFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        // enable CORS and disable CSRF
        http = http.cors(corsConfig -> corsConfig
                .configurationSource(configurationSource())
        ).csrf().disable();

        // set session management to stateless
        http = http
                .sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and();

        // set permissions on endpoints
        http.authorizeHttpRequests(authorize -> authorize
                .mvcMatchers("/api/redirect-url", "/api/
signin").permitAll()
                .mvcMatchers("/api/**").authenticated()
        );
```

3. We can add a simple JWT filter to intercept requests that need to verify tokens.

```
@Component
public class JwtTokenFilter extends OncePerRequestFilter {

    private final CasdoorAuthService casdoorAuthService;

    public JwtTokenFilter(CasdoorAuthService casdoorAuthService) {
        this.casdoorAuthService = casdoorAuthService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain chain)
            throws ServletException, IOException {
        // get authorization header and validate
        final String header =
request.getHeader(HttpHeaders.AUTHORIZATION);
        if (!StringUtils.hasText(header) ||
!header.startsWith("Bearer ")) {
            chain.doFilter(request, response);
            return;
        }

        // get jwt token and validate
        final String token = header.split(" ")[1].trim();

        // get user identity and set it on the spring security
context
        UserDetails userDetails = null;
        try {
            CasdoorUser casdoorUser =
casdoorAuthService.parseJwtToken(token);
            userDetails = new CustomUserDetails(casdoorUser);
        } catch (CasdoorAuthException exception) {
            logger.error("casdoor auth exception", exception);
            chain.doFilter(request, response);
            return;
```

When the user accesses the interface requiring authentication, `JwtTokenFilter` will obtain the token from the request header `Authorization` and verify it.

4. Next, we need to define a `Controller` to handle that when the user login to the `casdoor`, it will be redirected to the server and carry the `code` and `state`. The server needs to verify the user's identity from the `casdoor` and obtain the `token` through these two parameters.

```
@RestController
public class UserController {

    private static final Logger logger =
LoggerFactory.getLogger(UserController.class);

    private final CasdoorAuthService casdoorAuthService;

    // ...

    @PostMapping("/api/signin")
    public Result signin(@RequestParam("code") String code,
@RequestParam("state") String state) {
        try {
            String token = casdoorAuthService.getOAuthToken(code,
state);
            return Result.success(token);
        } catch (CasdoorAuthException exception) {
            logger.error("casdoor auth exception", exception);
            return Result.failure(exception.getMessage());
        }
    }

    // ...
}
```

# Step6. Try the demo!

First, you can try to access the frontend application through the browser. If you have not logged in, it will display a login button. Click the login button, and you will be redirected to the `casdoor` login page.

If you visit your root page,

Click the `Casdoor Login` button and the page will redirect to casdoor's login page.

After you log in, the page will redirect to `/`.

# Jenkins Plugin

Casdoor provides a plugin for users to login Jenkins. Here we will show you how to use Casdoor plugin for your Jenkins security.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`JENKINS_HOSTNAME`: Domain name or IP where Jenkins is deployed.

## Step1. Deploy Casdoor and Jenkins

Firstly, the Casdoor and Jenkins should be deployed.

After a successful deployment, you need to ensure:

1. Set Jenkins URL(Manage Jenkins → Configure System → Jenkins Location) to `JENKINS_HOSTNAME`.

2. Casdoor can be logged in and used normally.
3. Set Casdoor's `origin` value (conf/app.conf) to `CASDOOR_HOSTNAME`.

```
conf > app.conf
dbName = casdoor
redisEndpoint =
defaultStorageProvider =
isCloudIntranet = false
authState = "casdoor"
httpProxy = "127.0.0.1:10808"
verificationCodeTimeout = 10
initScore = 2000
logPostOnly = true
origin = "http://10.144.1.2:8000"
```

CASDOOR_HOSTNAME

# Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://JENKINS_HOSTNAME/securityRealm/finishLogin`

3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client`

`ID` and `Client secret` like the picture above, we will use them in next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Jenkins

Now, you can install Casdoor plugin from the market or by uploading its `jar` file.

After completing the installation, go to Manage Jenkins → Configure Global Security.

**Suggestion**: Back up the Jenkins `config.xml` file, and use it to recover in case of setup errors.

1. In Security Realm, select "Casdoor Authentication Plugin".
2. In Casdoor Endpoint, specify the `CASDOOR_HOSTNAME` noted above.

3. In Client ID, specify the `Client ID` noted above.
4. In Client secret, specify the `Client secret` noted above.
5. In JWT Public Key, specify the public key used to validate JWT token. You can find the public key in Casdoor by clicking `Cert` at the top. After clicking `edit` your application, you can copy your public key in the following page.

6. Organization Name and Application Name is optional. You can specify your organization and application to verify users in other organizations and applications. If they are empty, the plugin will use the default organization and application.
7. In the Authorization section, check “Logged-in users can do anything”. Disable “Allow anonymous read access”.
8. Click `save`.

Now, Jenkins will automatically redirect you to Casdoor for authentication.

# Jenkins OIDC

Casdoor can use OIDC protocol as IDP to connect various applications. Here we will use Jenkins as an example to show you how to use OIDC to connect to your applications.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`JENKINS_HOSTNAME`: Domain name or IP where Jenkins is deployed.

## Step1. Deploy Casdoor and Jenkins

Firstly, the Casdoor and Jenkins should be deployed.

After a successful deployment, you need to ensure:

1. Set Jenkins URL(Manage Jenkins → Configure System → Jenkins Location) to `JENKINS_HOSTNAME`.

2. Casdoor can be logged in and used normally.
3. Set Casdoor's `origin` value (conf/app.conf) to `CASDOOR_HOSTNAME`.

# Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://JENKINS_HOSTNAME/securityRealm/finishLogin`

3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Jenkins

First, we need to install OpenId Connect Authentication, Jenkins does not natively support OIDC.

After completing the installation, go to **Manage Jenkins → Configure Global Security**.

> **TIP**
>
> Back up the Jenkins `config.xml` file, and use it to recover in case of setup errors.

1. In Access Control, **Security Realm** select `Login with Openid Connect`.
2. In Client ID, specify the `Client ID` noted above.
3. In Client secret, specify the `Client secret` noted above.
4. In Configuration mode, select `Automatic configuration` and fill in http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration into Well-known configuration endpoint.

If your casdoor is deployed locally, you may need to select `Manual configuration` and input some information:

- Token server url: http://`CASDOOR_HOSTNAME`/api/login/oauth/access_token
- Authorization server url: http://`CASDOOR_HOSTNAME`/login/oauth/authorize
- UserInfo server url: http://`CASDOOR_HOSTNAME`/api/get-account
- Scopes: `address phone openid profile offline_access email`

5. Click **Advanced setting**, fill in the following:

- In User name field, specify `name`

- In Full name field, specify `displayName`
- In Email field, specify `email`

6. In the **Authorization** section, check “Logged-in users can do anything”. Disable “Allow anonymous read access”. You can configure more complex authorization later, for now check if OpenID actually works.

Log out of Jenkins, it should now redirect you to Casdoor for authentication.

# Jira

## Via Built-in SSO

Using OIDC protocol as IDP to connect various applications, like Jira

## Via miniOrange Plugin

Using OIDC protocol as IDP to connect various applications, like Jira

# Via Built-in SSO

This is a free method to connect casdoor, but your website must use https;

Casdoor can use OIDC protocol as IDP to connect various applications. Here is a Jira tutorial.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Jira_HOSTNAME`: Domain name or IP where Jira is deployed.

## Step1. Deploy Casdoor and Jira

Firstly, the Casdoor and Jira should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find Authentication methods:

3. Add a Configure and choose OpenId Connection signle sign-on in Authenication method

4. Find the redirect url:

5. Add a redirect url:

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Jira

1. We need continue to config our Configure in jira

## Edit existing configuration

Name *

casdoor

Use a unique name for this configuration.

Authentication method

OpenID Connect single sign-on

Users log in using OpenID Connect

### OpenID Connect settings

Issuer URL *

https://demo.casdoor.com

your casdoor url

The complete URL of the OpenID Provider. Needs to be unique.

Client ID *

642ec5d6779a2f0e879d

application client ID

The client identifier, as registered with the OpenID Provider.

Client secret *

••••••••••••••••••••••••••••••

application client secret

Change

Client secret is used in conjunction with the Client ID to authenticate the client application against the OpenID Provider.

Username mapping *

${preferred_username}

Used to map IdP claims to the username, e.g. ${sub}

Additional scopes

phone × email × address × profile ×

The default scope is 'openid'. Add more scopes if needed to obtain the username claim.

Redirect URL

https://test.v2tl.com/plugins/servlet/oidc/callback

Copy it to casdoor

Location where the client is sent to after successful account authentication.

Initiate login URL

https://test.v2tl.com/plugins/servlet/oidc/initiate-login

URL used for OpenID Provider-initiated login.

### Additional settings

The authorization, token, and user info endpoints will be filled automatically if your Identity provider offers this option. If not, you will be asked to provide this information.

- [x] Fill the data automatically from my chosen identity provider.

### JIT provisioning

Just-in-time user provisioning allows users to be created and updated automatically when they log in through SSO to Atlassian Data Center applications. Learn more.

- [ ] Create users on login to the application

### OpenID Connect behaviour

- [x] Remember user logins

  If checked, successful login history will be saved and users will be logged in automatically without the need for reauthentication.

### Login page settings

Decide if the IdP should be visible on login page and customize what the user will see on the button.

- [x] Show IdP on the login page

Login button text *

casdoor

The text is shown to the user on the login page. Remaining characters: 33.

Save configuration  Cancel

2. You can configure more complex authorization later, for now check if OpenID actually works.

You have temporary access to administrative functions. Drop access if you no longer require it. For more information, refer to the documentation.

Jira Software | Dashboards | Projects | Issues | Boards | Plans | Create | Search

# Administration

Search Jira admin

Applications | Projects | Issues | Manage apps | User management | Latest upgrade report | System

General configuration
Find more admin tools
Jira mobile app

SYSTEM SUPPORT
System info
Instrumentation
Monitoring
Database monitoring
Integrity checker
Logging and profiling
Scheduler details
Troubleshooting and support tools
Clean up
Audit log
Clustering

SECURITY
Project roles
Global permissions

## Authentication methods

Add configuration

Manage how users authenticate. Save authentication configurations using SAML, OpenID Connect, or Crowd as the identity provider. Learn more about using multiple identity providers.

**Make authentication safer**

Authenticating with username and password is less secure than through single sign-on. Now that you've configured the latter, consider disabling product login form and basic authentication.

Communicate this change to your users.

How to disable · Dismiss

### Login options

| Name | Type | Last updated | Show on login page | Actions |
|---|---|---|---|---|
| Username and password | Product login form | Never | ✓ | ••• |
| casdoor | OpenID Connect | 26 April 2023 7:20 PM | ✓ | ••• |

### Authentication on API calls

Allow basic authentication on API calls.

You can use personal access tokens as a safer alternative method of authentication. See Using personal access tokens.

# Via miniOrange Plugin

This is a tutorial on using miniOrange to connect casdoor and jira

Casdoor can use OIDC protocol as IDP to connect various applications. Here is a Jira tutorial.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Jira_HOSTNAME`: Domain name or IP where Jira is deployed.

## Step1. Deploy Casdoor and Jira

Firstly, the Casdoor and Jira should be deployed.

After a successful deployment, you need to ensure:

1. Set Jira URL(Plans → Administration → System → General configuration) to `Jira_HOSTNAME`.

2. Casdoor can be logged in and used normally.

3. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

# Step2. Configure Casdoor application and Jira

1. Create or use an existing Casdoor application.
2. You should install a miniOrange app to support OAuth.You can find this app in Plans->Administration->Find new apps->search

3. Set `Selected Application` to Custom OpenId
4. Find a redirect url:

5. Add a redirect url:

6. You should config this app

- `Token server url`: http://`CASDOOR_HOSTNAME`/api/login/oauth/access_token
- `Authorization server url`: http://`CASDOOR_HOSTNAME`/login/oauth/authorize
- `UserInfo server url`: http://`CASDOOR_HOSTNAME`/api/get-account
- `Scopes`: `address phone openid profile offline_access email`

Open your favorite browser and visit: http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration, you will see the OIDC configure of Casdoor.

Log out of Jira, and test SSO.

Jira Software
Dashboards
Search
Log In
Welcome to JIRA
Username
Password
Remember my login on this computer
Not a member? To request an account, please contact your Jira administrators.
Log In
Use OAuth Login
Can't access your account?
Atlassian Jira Project Management Software · About Jira · Report a problem
Powered by a free Atlassian Jira evaluation license. Please consider purchasing it today.
ATLASSIAN

# Confluence

Casdoor can use OIDC protocol as IDP to connect various applications. Here we will use Confluence as an example to show you how to use OIDC to connect to your applications.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Confluence_HOSTNAME`: Domain name or IP where Confluence is deployed.

## Step1. Deploy Casdoor and Confluence

Firstly, the Casdoor and Confluence should be deployed.

After a successful deployment, you need to ensure:

1. Set Confluence URL to `Confluence_HOSTNAME`.

2. Casdoor can be logged in and used normally.
3. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

# Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find a redirect url:

3. Add a redirect url:

4. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Confluence

1. You should install a miniOrange app to support OAuth.You can find this app in

2. You should config this app

3. Set `Selected Application` to Custom OpenId
4. You can find Client Id and Client Secret in Casdoor application page.

- `Token server url`: http://**`CASDOOR_HOSTNAME`**/api/login/oauth/access_token
- `Authorization server url`: http://**`CASDOOR_HOSTNAME`**/login/oauth/authorize
- `UserInfo server url`: http://**`CASDOOR_HOSTNAME`**/api/get-account
- `Scopes`: `address phone openid profile offline_access email`

1. You can configure more complex authorization later, for now check if OpenID actually works.

Log out of Confluence, and test SSO.

Log in - Confluence
localhost:8090/login.action?language=en_GB
Confluence
Log in
Username
Password
Remember me
Log in
Use OAuth Login
Forgot your password?
EVALUATION LICENSE Are you enjoying Confluence? Please consider purchasing it today.
Powered by Atlassian Confluence 8.1.1 · Report a bug · Atlassian News
ATLASSIAN
localhost:8090/plugins/servlet/oauth/auth?return_to=http://localhost:8090&idp=4f6b30c1-eba8-4b89-ac02-4a4b7a137b97&idp=4f6b30c1-eba8-4b89-ac02-4a4b7a137b97

# RuoYi

Casdoor can simply connect to RuoYi-cloud.

## Step 1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor-based login page in your own app with the following steps.

## Step 2. Configure Casdoor

Configure casdoor can refer to casdoor(Configure casdoor's browser better not use one browser as your develop browser).

You also should configure an organization, an application and the Synchronizer. You also can refer to casdoor.

Some points needing attention:

1. The table columns in edit syncer:

2. The password type in edit organization:

3. You also should open soft deletion.

# Step 3. Reform your front-end

## 3.1 jump to casdoor's login page

We can use front-end sdk, take vue-sdk as an example here. After you init vue-sdk, you can get casdoor login page url by getSigninUrl().

You can link it with the way you like and you can delete some ruoyi-cloud original code which you have no further use, such as original account and password el-input.

## 3.2 Accept the code and state which return by casdoor

After we login in successfully by casdoor, casdoor sends the code and state to the page that we set up. We can get the code and state with function create.

```
created() {
    let url = window.document.location.href//get url
```

For RuoYi-Cloud, we just change its original method which sends account and password to send code and state. Therefore, it just changes what is sent to the back end, relative to the original login.

# Step 4. Reform your back-end

## 4.1 Accept the code and state which return by front-end

```
@PostMapping("login")
public R<?> callback(@RequestBody CodeBody code) {//we should define
a CodeBody entity which have code and state
    String token = casdoorAuthService.getOAuthToken(code.getCode(),
code.getState());
    CasdoorUser casdoorUser =
casdoorAuthService.parseJwtToken(token);
    if(casdoorUser.getName()!=null){
        String casdoorUserName = casdoorUser.getName();

if(sysLoginService.getUserByCasdoorName(casdoorUserName)==null){//if
database haven't this user
            // add this user into database
            sysLoginService.casdoorRegister(casdoorUserName);
        }
    }
    LoginUser userInfo =
sysLoginService.casdoorLogin(casdoorUser.getName());//get this
user's information by database
    return R.ok(tokenService.createToken(userInfo));
}
```

In this method, we use casdoor-SpringBoot-sdk method and slightly modified RuoYi-Cloud Method.

For example, RuoYi-Cloud original register with account and password, I change the register with account like casdoorRegister.

I also add a method to execute whether this account exists like getUserByCasdoorName and change execute userinfo with account and password to with account.

It's easy, because we only need to delete the part of checking password.

# Step 5. Summary

## 5.1 front-end

- We need to delete original login and register.
- We also need to accept code and state and send them to back-end.

## 5.2 back-end

RuoYi back-end has perfect login and registration function. We just need to change a little, so it is very convenient.

# Step 6. Detailed steps

1. Deploy and configure casdoor. We must take care of the organization's password type which should choose bcrypt because RuoYi-Cloud's password type is bcrypt.

2. We should use casdoor syncers to copy database users to your casdoor organization. This step can make the original account import to casdoor.

3. After we deployed casdoor, we should change front-end. We should close RuoYi check code

```
// checkcode switch
captchaEnabled: false,
// register switch
register: true,
```

Note that RuoYi-Cloud captcha needs to be close in nacos again. Note that RuoYi-Cloud open registration function requires changing sys.account.registerUser to true.

4. We should add button jump to casdoor and change data's loginForm

```
<a href="http://localhost:7001/login/oauth/authorize?client_id=d509b6b3edc8a3d4cce9&response_type=code&redirect_uri=http%3A%2F%2Flocalhost
">casdoor</a>
```

```
loginForm: {
  code: "",
  state: ""
},
```

Here I write url, you can get url by casdoor-vue-sdk or casdoor-SpringBoot-sdk.

5. Because we don't use the original login, we should delete the method about cookie and checkcode method.

   So the new create function:

```
created() {
    let url = window.document.location.href//get url
    let u = new URL(url);
    this.loginForm.code = u.searchParams.get('code')//get code and state
    this.loginForm.state = u.searchParams.get('state')
    if(this.loginForm.code!=null&&this.loginForm.state!=null){//if code and state is null, execute handleLogin
      this.handleLogin()
    }
}
```

6. In fact, we just need to change the parameter we send to back-end and delete the function we don't need, we don't need to change anything else.

```
handleLogin() {
  console.log("进入handleLogin")
      this.$store.dispatch("Login", this.loginForm).then(() => {
        this.$router.push({ path: this.redirect || "/" }).catch(()=>{});
      }).catch(() => {
        this.loading = false;
        if (this.captchaEnabled) {
          this.getCode();
          console.log(this.getCode)
        }
      });
  }
}
```

```
Login({ commit }, userInfo) {
  const code = userInfo.code
  const state = userInfo.state
  return new Promise((resolve, reject) => {
    login(code, state).then(res => {
      console.log("LOGIN")
      let data = res.data
      setToken(data.access_token)
      commit('SET_TOKEN', data.access_token)
      setExpiresIn(data.expires_in)
      commit('SET_EXPIRES_IN', data.expires_in)
      resolve()
    }).catch(error => {
      reject(error)
    })
  })
},
```

```
export function login(code, state) {
  return request({
    url: '/auth/login',
    headers: {
      isToken: false
    },
    method: 'post',
    data: {code, state}
  })
}
```

7. Import dependency in back-end.

pom.xml

```
<dependency>
    <groupId>org.casbin</groupId>
    <artifactId>casdoor-spring-boot-starter</artifactId>
    <version>1.2.0</version>
</dependency>
```

You also need to configure casdoor in resource. 8. Callback function is defined as the redirect function. I change some methods in sysLoginService slightly. I delete the check password step because we don't need it.

```
@PostMapping("login")
public R<?> callback(@RequestBody CodeBody code) {//we should define
a CodeBody entity which have code and state
    String token = casdoorAuthService.getOAuthToken(code.getCode(),
code.getState());
    CasdoorUser casdoorUser =
casdoorAuthService.parseJwtToken(token);
    if(casdoorUser.getName()!=null){
        String casdoorUserName = casdoorUser.getName();

if(sysLoginService.getUserByCasdoorName(casdoorUserName)==null){//if
database haven't this user
            // add this user into database
            sysLoginService.casdoorRegister(casdoorUserName);
        }
    }
    LoginUser userInfo =
sysLoginService.casdoorLogin(casdoorUser.getName());//get this
user's information by database
    return R.ok(tokenService.createToken(userInfo));
}
```

9. SysLoginService's new method

```
public LoginUser casdoorLogin(String username){
    // execute user
    R<LoginUser> userResult =
remoteUserService.getUserInfo(username, SecurityConstants.INNER);
    if (R.FAIL == userResult.getCode())
    {
        throw new ServiceException(userResult.getMsg());
    }

    if (StringUtils.isNull(userResult) ||
StringUtils.isNull(userResult.getData()))
    {
        recordLogService.recordLogininfor(username,
Constants.LOGIN_FAIL, "this user is not exist");
        throw new ServiceException("user" + username + " is not
exist");
    }
    LoginUser userInfo = userResult.getData();
    SysUser user = userResult.getData().getSysUser();
    if (UserStatus.DELETED.getCode().equals(user.getDelFlag()))
    {
        recordLogService.recordLogininfor(username,
Constants.LOGIN_FAIL, "sorry, your account was deleted");
        throw new ServiceException("sorry, your account" +
username + " was deleted");
    }
    if (UserStatus.DISABLE.getCode().equals(user.getStatus()))
    {
        recordLogService.recordLogininfor(username,
Constants.LOGIN_FAIL, "your account is disabled, you can contact
admin ");
         throw new ServiceException(sorry, your account" +
username + " is disabled");
    }
    recordLogService.recordLogininfor(username,
Constants.LOGIN_SUCCESS, "login successfully");
    return userInfo;
```

```java
public String getUserByCasdoorName(String casdoorUsername){
    R<LoginUser> userResult =
remoteUserService.getUserInfo(casdoorUsername,
SecurityConstants.INNER);
    if (StringUtils.isNull(userResult) ||
StringUtils.isNull(userResult.getData()))
    {
        //if this user is not in RuoYi-Cloud database and casdoor
have this user, we should create this user in database
        return null;
    }
    String username =
userResult.getData().getSysUser().getUserName();
    return username;
}
```

```java
public void casdoorRegister(String username){
    if (StringUtils.isAnyBlank(username))
    {
        throw new ServiceException("User must fill in");
    }
    SysUser sysUser = new SysUser();
    sysUser.setUserName(username);
    sysUser.setNickName(username);
    R<?> registerResult =
remoteUserService.registerUserInfo(sysUser, SecurityConstants.INNER);
    System.out.println(registerResult);
    if (R.FAIL == registerResult.getCode())
    {
        throw new ServiceException(registerResult.getMsg());
    }
    recordLogService.recordLogininfor(username, Constants.REGISTER,
"register successfully");
}
```

# Pulsar-manager

Casdoor can simply connect to Pulsar-manager.

Because the code for connecting casdoor has been added in Pulsar-manager, we just need to configure application.yml in back-end and open front switch.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a Casdoor-based login page in your own app with the following steps.

## Step2. Configure Casdoor

Configure casdoor can refer to casdoor(Configure casdoor's browser better not use one browser as your develop browser).

You also should configure an organization and an application. You also can refer to casdoor.

## step2.1 you should create an organization

## step2.2 you should create an application

# Step3. Open Pulsar-manager front-end

# switch

Open this switch to make code and state send to back-end.

This switch in the Line 80 of pulsar-manager/front-end/src/router/index.js

```
-   // mode: 'history', // require service support
+   mode: 'history', // require service support
```

# Step4. Configure back-end code

You should configure casdoor's Configuration in the Line 154 of pulsar-manager/src/main/resources/application.properties

```
casdoor.endpoint = http://localhost:8000
casdoor.clientId = <client id in previous step>
casdoor.clientSecret = <client Secret in previous step>
casdoor.certificate=<client certificate in previous step>
casdoor.organizationName=pulsar
casdoor.applicationName=app-pulsar
```

# ShenYu

ShenYu has casdoor plugin to use casdoor

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application
2. Add Your redirect url

3. On the certificate editing page, you can see your `Certificate`

# Step3. Use casdoor plugin in shenyu

## 1. Config casdoor plugin in shenyu

note: because the shenyu only have Single line input box so we need add \n in

every line of cert.

Certificate ⓘ : Copy certificate Download certificate


```
-----BEGIN CERTIFICATE-----\n
MIIE+TCCAuGgAwIBAgIDAeJAMA0GCSqGSIb3DQEBCwUAMDYxHTAbBgNVBAoTFENh\n
c2Rvb3IgT3JnYW5pemF0aW9uMRUwEwYDVQQDEwxDYXNkb29yIENIcnQwHhcNMjEx\n
MDE1MDgxMTUyWhcNNDExMDE1MDgxMTUyWjA2MR0wGwYDVQQKExRDYXNkb29yIE9y\n
Z2FuaXphdGlvbjEVMBMGA1UEAxMMQ2FzZG9vciBDZXJ0MIICIjANBgkqhkiG9w0B\n
AQEFAAOCAg8AMIICCgKCAgEAslnpb5E1/ym0f1RfSDSSE8IR7y+Iw+RJjI74e5ej\n
rq4b8zMYk7HeHCyZr/hmNEwEVXnhXu1P0mBeQ5ypp/QGo8vgEmjAETNmzkl1NjOQ\n
CjCYwUrasO/f/Mnl1C0j13vx6mV1kHZjSrKsMhYY1vaxTEP3+VB8Hjg3MHFWrb07\n
uvFMCJe5W8+0rKErZCKTR8+9VB3janeBz//zQePFVh79bFZate/hLirPK0Go9P1g\n
OvwloC1A3sarHTP4Qm/LQRt0rHqZFybdySpyWAQvhNaDFE7mTstRSBb/wUjNCUBD\n
PTSLVjC04WIISf6Nkfx0Z7KvmbPstSj+btvcqsvRAGtvdsB9h62Kptjs1Yn7GAuo\n
I3qt/4zoKbiURYxkQJXlvwCQsEftUuk5ew5zuPSIDRLoLByQTLbx0JqLAFNfW3g/\n
pzSDjgd/60d6HTmvbZni4SmjdyFhXCDb1Kn7N+xTojnfaNkwep2REV+RMc0fx4Gu\n
hRsnLsmkmUDeylZ9aBL9oj11YEQfM2JZEq+RVtUx+wB4y8K/tD1bcY+IfnG5rBpw\n
IDpS262boq4SRSvb3Z7bB0w4ZxvOfJ/1VLoRftjPbLlf0bhfr/AeZMHplKOXvfz4\n
yE+hqzi68wdF0VR9xYc/RbSAf7323OsjYnjjEgInUtRohnRgCpjIk/Mt2Kt84Kb0\n
wn8CAwEAAaMQMA4wDAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQsFAAOCAgEAn2If\n
DKkLX+F1vKRO/5gJ+Plr8P5NKuQkmwH97b8CS2gS1phDyNgIc4/LSdzuf4Awe6ve\n
C06lVdWSIis8UPUPdjmT2uMPSNjwLxG3QsrimMURNwFILTfRem/heJe0Zgur9J1M\n
8haawdSdJjH2RgmFoDeE2r8NVRfhbR8KnCO1ddTJKuS1N0/irHz21W4jt4rxzCvl\n
2nR42Fybap3O/g2JXMhNNROwZmNjgpsF7XVENCSuFO1jTywLaqjuXCg54IL7XVLG\n
omKNNNcc8h1FCeKj/nnbGMhodnFWKDTsJcbNmcOPNHo6ixzqMy/Hqc+mWYv7maAG\n
Jtevs3qgMZ8F9Qzr3HpUc6R3ZYYWDY/xxPisuKftOPZgtH979XC4mdf0WPnOBLqL\n
2DJ1zaBmjiGJoIvb7XNVKcUfDXYw85ZTZQ5b9cIl4e+6bmyWqQltIwt+Ati/uFEV\n
XzCj70B4IALX6xau1kLEpV9O1GERizYRz5P9NJNA7KoO5AVMp9w0DQTkt+LbXnZE\n
HHnWKy8xHQKZF9sR7YBPGLs/Ac6tviv5Ua15OgJ/8dLRZ/veyFfGo2yZsI+hKVU5\n
nCCJHBcAyFnm1hdvdwEdH33jDBjNB6ciotJZrf/3VYaIWSaIADosHAgMWfXuWP+h\n
8XKXmzIxuHbTMQYtZPDgspS5aK+S4Q9wb8RRAYo=\n
-----END CERTIFICATE-----
```


here not need add \n

You can copy it and paste it on the certificate of shenyu casdoor config.

You don't need save it in casdoor certificate editing page,because it just for copying.

# 2. Confing shenyu casdoor's plugin

You can config what you need to use casdoor config

# 3. Get the service and use

## 3.1 Visit the Web directly like

## 3.2 Use casdoor login like this

hi! null! I'm Shenyu-Gateway System. Welcome!

3.3 Carry token in Headers,you also can visit it

3.4 It also can save name,id and organization in Headers so that you can use them in next time

# ShardingSphere

shardingsphere-elasticjob-ui have integrated Casdoor. We can use it after config it.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a casdoor based login page in your own app with the following steps.

## Step2. Configure Casdoor application and configure application in ShardingSphere

1.Create or use an existing Casdoor application

RedirectURLs is depend on what url you need redirect.The selected data will use in next.

2.On the certificate editing page, you can see your `Certificate`

Certificate ⓘ : Copy certificate Download certificate Private


```
-----BEGIN CERTIFICATE-----
MIIE+TCCAuGgAwIBAgIDAeJAMA0GCSqGSIb3DQEBCwUAMDYxHTAbBgNVBAoTFENh
c2Rvb3IgT3JnYW5pemF0aW9uMRUwEwYDVQQDEwxDYXNkb29yIENlcnQwHhcNMjEx
MDE1MDgxMTUyWhcNNDExMDE1MDgxMTUyWjA2MR0wGwYDVQQKExRDYXNkb29yIE9y
Z2FuaXphdGlvbjEVMBMGA1UEAxMMQ2FzZG9vciBDZXJ0MIICIjANBgkqhkiG9w0B
AQEFAAOCAg8AMIICCgKCAgEAslnpb5E1/ym0f1RfSDSSE8IR7y+Iw+RJjl74e5ej
rq4b8zMYk7HeHCyZr/hmNEwEVXnhXu1P0mBeQ5ypp/QGo8vgEmjAETNmzkI1NjOQ
CjCYwUrasO/f/MnI1C0j13vx6mV1kHZjSrKsMhYY1vaxTEP3+VB8Hjg3MHFWrb07
uvFMCJe5W8+0rKErZCKTR8+9VB3janeBz//zQePFVh79bFZate/hLirPK0Go9P1g
OvwIoC1A3sarHTP4Qm/LQRt0rHqZFybdySpyWAQvhNaDFE7mTstRSBb/wUjNCUBD
PTSLVjC04WIISf6Nkfx0Z7KvmbPstSj+btvcqsvRAGtvdsB9h62Kptjs1Yn7GAuo
I3qt/4zoKbiURYxkQJXIvwCQsEftUuk5ew5zuPSIDRLoLByQTLbx0JqLAFNfW3g/
pzSDjgd/60d6HTmvbZni4SmjdyFhXCDb1Kn7N+xTojnfaNkwep2REV+RMc0fx4Gu
hRsnLsmkmUDeyIZ9aBL9oj11YEQfM2JZEq+RVtUx+wB4y8K/tD1bcY+IfnG5rBpw
IDpS262boq4SRSvb3Z7bB0w4ZxvOfJ/1VLoRftjPbLIf0bhfr/AeZMHpIKOXvfz4
yE+hqzi68wdF0VR9xYc/RbSAf7323OsjYnjjEgInUtRohnRgCpjIk/Mt2Kt84Kb0
wn8CAwEAAaMQMA4wDAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQsFAAOCAgEAn2If
DKkLX+F1vKRO/5gJ+Plr8P5NKuQkmwH97b8CS2gS1phDyNgIc4/LSdzuf4Awe6ve
C06IVdWSIis8UPUPdjmT2uMPSNjwLxG3QsrimMURNwFILTfRem/heJe0Zgur9J1M
8haawdSdJjH2RgmFoDeE2r8NVRfhbR8KnCO1ddTJKuS1N0/irHz21W4jt4rxzCvl
2nR42Fybap3O/g2JXMhNNROwZmNjgpsF7XVENCSuFO1jTywLaqjuXCg54IL7XVLG
omKNNNcc8h1FCeKj/nnbGMhodnFWKDTsJcbNmcOPNHo6ixzqMy/Hqc+mWYv7maAG
Jtevs3qgMZ8F9Qzr3HpUc6R3ZYYWDY/xxPisuKftOPZgtH979XC4mdf0WPnOBLqL
2DJ1zaBmjiGJoIvb7XNVKcUfDXYw85ZTZQ5b9cIl4e+6bmyWqQltlwt+Ati/uFEV
XzCj70B4lALX6xau1kLEpV9O1GERizYRz5P9NJNA7KoO5AVMp9w0DQTkt+LbXnZE
HHnWKy8xHQKZF9sR7YBPGLs/Ac6tviv5Ua15OgJ/8dLRZ/veyFfGo2yZsI+hKVU5
nCCJHBcAyFnm1hdvdwEdH33jDBjNB6ciotJZrf/3VYalWSaIADosHAgMWfXuWP+h
8XKXmzlxuHbTMQYtZPDgspS5aK+S4Q9wb8RRAYo=
-----END CERTIFICATE-----
```


## 3.Configure application in ShardingSphere

First we need find the application.properties we need configure

Second we need copy the data in Casdoor application and paste them into application.

```
casdoor.endpoint=http://localhost:7001
casdoor.client-id=3ed79fa530645fbd3653
casdoor.client-secret=54633c82b7796a4332c6976864c6c16bc3b05556
casdoor.certificate=\
-----BEGIN CERTIFICATE-----\n\
MIIE+TCCAuGgAwIBAgIDAeJAMA0GCSqGSIb3DQEBCwUAMDYxHTAbBgNVBAoTFENh\n\
c2Rvb3IgT3JnYW5pemF0aW9uMRUwEwYDVQQDEwxDYXNkb29yIENlcnQwHhcNMjEx\n\
MDE1MDgxMTUyWhcNNDExMDE1MDgxMTUyWjA2MR0wGwYDVQQKExRDYXNkb29yIE9y\n\
Z2FuaXphdGlvbjEVMBMGA1UEAxMMQ2FzZG9vciBDZXJ0MIICIjANBgkqhkiG9w0B\n\
AQEFAAOCAg8AMIICCgKCAgEAsInpb5E1/ym0f1RfSDSSE8IR7y+lw+RJjI74e5ej\n\
rq4b8zMYk7HeHCyZr/hmNEwEVXnhXu1P0mBeQ5ypp/QGo8vgEmjAETNmzkI1NjOQ\n\
CjCYwUrasO/f/MnI1C0j13vx6mV1kHZjSrKsMhYY1vaxTEP3+VB8Hjg3MHFWrb07\n\
uvFMCJe5W8+0rKErZCKTR8+9VB3janeBz//zQePFVh79bFZate/hLirPK0Go9P1g\n\
OvwIoC1A3sarHTP4Qm/LQRt0rHqZFybdySpyWAQvhNaDFE7mTstRSBb/wUjNCUBD\n\
PTSLVjC04WllSf6Nkfx0Z7KvmbPstSj+btvcqsvRAGtvdsB9h62Kptjs1Yn7GAuo\n\
I3qt/4zoKbiURYxkQJXIvwCQsEftUuk5ew5zuPSlDRLoLByQTLbx0JqLAFNfW3g/\n\
pzSDjgd/60d6HTmvbZni4SmjdyFhXCDb1Kn7N+xTojnfaNkwep2REV+RMc0fx4Gu\n\
hRsnLsmkmUDeyIZ9aBL9oj11YEQfM2JZEq+RVtUx+wB4y8K/tD1bcY+IfnG5rBpw\n\
IDpS262boq4SRSvb3Z7bB0w4ZxvOfJ/1VLoRftjPbLIf0bhfr/AeZMHpIKOXvfz4\n\
yE+hqzi68wdF0VR9xYc/RbSAf7323OsjYnjjEgInUtRohnRgCpjIk/Mt2Kt84Kb0\n\
wn8CAwEAAaMQMA4wDAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQsFAAOCAgEAn2lf\n\
DKkLX+F1vKRO/5gJ+Plr8P5NKuQkmwH97b8CS2gS1phDyNgIc4/LSdzuf4Awe6ve\n\
C06lVdWSIis8UPUPdjmT2uMPSNjwLxG3QsrimMURNwFlLTfRem/heJe0Zqur9J1M\n\
8baawdSdJjH2RgmFoDeE2r8NVRfhbR8KnCO1ddTJKuS1N0/irHz21W4jt4rxzCvl\n\
2nR42Fybap3O/g2JXMhNNROwZmNjgpsF7XVENCSuFO1jTywLaqjuXCg54IL7XVLG\n\
omKNNNcc8h1FCeKj/nnbGMhodnFWKDTsJcbNmcOPNHo6ixzqMy/Hqc+mWYv7maAG\n\
Jtevs3qgMZ8F9Qzr3HpUc6R3ZYYWDY/xxPisuKftOPZgtH979XC4mdf0WPnOBLqL\n\
2DJ1zaBmjiGJolvb7XNVKcUfDXYw85ZTZQ5b9clI4e+6bmyWqQItlwt+Ati/uFEV\n\
XzCj70B4lALX6xau1kLEpV9O1GERizYRz5P9NJNA7KoO5AVMp9w0DQTkt+LbXnZE\n\
HHnWKy8xHQKZF9sR7YBPGLs/Ac6tviv5Ua15OgJ/8dLRZ/veyFfGo2yZsI+hKVU5\n\
nCCJHBcAyFnm1hdvdwEdH33jDBjNB6ciotJZrf/3VYaIWSalADosHAgMWfXuWP+h\n\
8XKXmzlxuHbTMQYtZPDqspS5aK+S4Q9wb8RRAYo=\n\
-----END CERTIFICATE-----
casdoor.organization-name=ShardingSphere
casdoor.application-name=ShardingSphere
```

4.Test it

ShardingSphere
username
password
Login
Login with Casdoor
Copyright © The Apache Software Foundation, Licensed under the Apache License 2

# Apache IoTDB

Casdoor can simply connect to Apache IoTDB.

Because the code for connecting casdoor has been added in Apache IoTDB Web Workbench, we just need to configure application.yml in back-end and open front switch.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on **http://localhost:8000**.
- Open your favorite browser and visit **http://localhost:7001**, you will see the login page of Casdoor.
- Input `admin` and `123` to test login functionality is working fine.

Then you can quickly implement a Casdoor-based login page in your own app with the following steps.

## Step2. Configure Casdoor

Configure casdoor can refer to casdoor(Configure casdoor's browser better not use one browser as your develop browser).

You also should configure an organization and an application. You also can refer to casdoor.

## 2.1 you should create an organization

## 2.2 you should create an application

# Step3. Open Apache IoTDB Web Workbench front-end switch

Open this switch to make code and state send to back-end.

This switch in the iotdb-web-workbench/fronted/.env

# Step4. Configure back-end code

You should configure casdoor's Configuration in the iotdb-web-workbench/backend/src/main/resources/application.properties

```
casdoor.endpoint = http://localhost:8000
casdoor.clientId = <client id in previous step>
casdoor.clientSecret = <client Secret in previous step>
casdoor.certificate=<client certificate in previous step>
casdoor.organizationName=IoTDB
casdoor.applicationName=app-IoTDB
```

# Result

IoTDB WorkBench
IoTDB WorkBench
Welcome To IoTDB WorkBench
Account
Please Input Account
Password
Forget Password?
Sign In
Sign In With Casdoor

# Apache DolphinScheduler

Casdoor is one of the supported login method for Apache DolphinScheduler.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

You can refer to the Casdoor official documentation for the Server Installation.

After a successful deployment, you need to ensure:

- The Casdoor server is successfully running on http://localhost:8000.
- Open your favorite browser and visit http://localhost:7001, you will see the login page of Casdoor.
- Input admin and 123 to test login functionality is working fine.

Then you can quickly implement a Casdoor based login page in your own app with the following steps.

## Step2. Configure Casdoor Application

1. Create or use an existing Casdoor application.
2. Add Your redirect url (You can see more details about how to get redirect url in the next section)

3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above. We will use them in next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Dolphinscheduler

> dolphinscheduler-api/src/main/resources/application.yaml

```yaml
security:
  authentication:
    # Authentication types (supported types:
PASSWORD,LDAP,CASDOOR_SSO)
    type: CASDOOR_SSO
casdoor:
  # Your Casdoor server url
  endpoint:
  client-id:
  client-secret:
  # The certificate may be multi-line, you can use `|-` for ease
  certificate:
  # Your organization name added in Casdoor
  organization-name:
  # Your application name added in Casdoor
  application-name:
  # Doplhinscheduler login url
  redirect-url: http://localhost:5173/login
```

Now, Dolphinschduler will automatically redirect you to Casdoor for authentication.

en_US
DolphinScheduler
SSO Login

# FireZone

Casdoor can use OIDC protocol as IDP to connect various applications. Here we will use FireZone as an example to show you how to use OIDC to connect to your applications.

## Step 1. Deploy Casdoor and FireZone

Firstly, the Casdoor and FireZone should be deployed.

After a successful deployment, you need to ensure:

1. Set FireZone URL(Sigin → Security → Add OpenID Connect Provider) to FIREZONE_HOSTNAME.

2. Casdoor can be logged in and used normally.

3. `CASDOOR_HOSTNAME`: http://localhost:8000. If you deploy Casdoor using default `app.conf`.

# Step 2. Configure Casdoor application

1. Create or use an existing Casdoor application.

2. Add a redirect url:

For example, the Configid in the FireZone Provider is TEST, so the redirect URL should be `http://[FIREZONE_HOST]/auth/oidc/[PROVIDER_CONFIG_ID]/callback/`

Open your favorite browser and visit: `http://[CASDOOR_HOSTNAME]/.well-known/openid-configuration`, you will see the OIDC configure of Casdoor.

3. Configure FireZone, Security → Add OpenID Connect Provider

- `Discovery Document URI`: FireZone Provider Discovery Document URI should be `https://[CASDOOR_HOST]/.well-known/openid-configuration`
- `Scopes`: `openid email profile`

- `ConfigID`: ConfigID should be the PROVIDER_COONFIG_ID of the redirect URL and should correspond to casdoor redirect URL
- `Auto create users`: Successful login will automatically create a user

# Log out of FireZone, and test SSO.

# CloudFoundry

Before the integration, we need to deploy Casdoor locally.

Then we can quickly implement a Casdoor-based login page in our own app with the following steps.

## Step1. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://CASDOOR_HOSTNAME/login`

3. Copy the client ID, we will need it in the following steps.

## Step2. Add user in Casdoor

Now you have the application, but not a user. That means you need to create a user and assign the role.

Go to the “Users” page and click on “Add user” in the top right corner. That opens a new page where you can add the new user.

Save the user after adding a username and adding the organisation CloudFoundry(other details are optional).

Now you need to set up a password for your user, which you can do by clicking manage your password.

Choose a password for your user and confirm it.

## Step3. Build CloudFoundry App

Start the CloudFoundry by follow.

- $git clone git://github.com/cloudfoundry/uaa.git
- $ cd uaa
- $./gradlew run

# Step4. Integrate Casdoor

Now open another command line and input :

- $ curl 'http://localhost/oauth/authorize?response_type=token&client_id=app&scope=openid&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Fapp%2F' -i -X GET \ -H 'Accept: application/x-www-form-urlencoded'

we have already got the client_id and redirect_uri before, we input these parameters.

| Parameter | Type | Constraints | Description |
| --- | --- | --- | --- |
| response_type | String | Required | Space-delimited list of response types. Here, `token`, i.e. an access token |
| client_id | String | Required | a unique string representing the registration information provided by the client |
| scope | String | Optional | requested scopes, space-delimited |
| redirect_uri | String | Optional | redirection URI to which the authorization server will send the user-agent back once access is granted (or denied), optional if pre-registered by the client |

execute the command, we can get the result below, which means that we have integrated Casdoor with CloudFoundry successfully.

```
HTTP/1.1 302 Found
Content-Security-Policy: script-src 'self'
Strict-Transport-Security: max-age=31536000
Set-Cookie: X-Uaa-Csrf=09mMqMDhcwHGLMufnB4YA1; Path=/; Max-Age=86400; Expires=Fri, 5 May 2023 14:53:54 GMT; HttpOnly; SameSite=Lax
Cache-Control: no-store
Content-Language: en
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Location: http://localhost:8080/app/#token_type=bearer&access_token=eyJhbGciOiJIUzI1NiIsImprdSI6Imh0dHBzOi8vbG9jYWxob3N0OjgwODAvdWFhL3Rva
```

Built-in Organization
localhost:8000/login
Casdoor
username, Email or phone
Password
Auto sign in
Forgot password?
Sign In
No account? sign up now
Powered by Casdoor

# JavaScript

## WeChat MiniProgram

Using Casdoor in WeChat MiniProgram

# WeChat MiniProgram

> INFO
>
> Casdoor supports WeChat Mini Program after version 1.41.0

## Introduction

Since WeChat Mini Program does not support standardized OAuth, it cannot jump to the self-host Casdoor webpage for login. Therefore, the process of using Casdoor for WeChat Mini Program is different from that of ordinary programs.

This document will talk about how to access Casdoor to WeChat Mini Program. You can find the example on GitHub here: casdoor-wechat-miniprogram-example. More detailed information can be found in the WeChat Mini Program login document.

> The following are some of the names in the configuration:
>
> `CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed. e.g., `https://door.casbin.com`.

## Step1. Deploy Casdoor

Firstly, the Casdoor should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. Set Casdoor's `origin` value (conf/app.conf) to `CASDOOR_HOSTNAME`.

```
conf > app.conf
dbName = casdoor
redisEndpoint =
defaultStorageProvider =
isCloudIntranet = false
authState = "casdoor"
httpProxy = "127.0.0.1:10808"
verificationCodeTimeout = 10
initScore = 2000
logPostOnly = true
origin = "http://10.144.1.2:8000"
                CASDOOR_HOSTNAME
```

# Step2. Configure Casdoor application

1. Create a wechat idp in casdoor and fill your `APPID` and `APPSECRET` given to you by WeChat Mini Program develop platform:

2. Create or use an existing Casdoor application.
3. Add the idp added above to the application you want to use.

> **TIPS**
>
> For convenience, casdoor will read the first WeChat type idp in the application as the WeChat Mini Program idp by default.
>
> So if you want to use the WeChat Mini Program in this app, don't add multiple WeChat type idp in one app.

# Step3. Write WeChat MiniProgram code

WeChat Mini Program provides an API to login internally and gets the Code. All you need to do is to send this Code to Casdoor. Casdoor will use this Code to get some information from WeChat server (such as OpenID, SessionKey, etc.).

The following code shows how to accomplish the above process:

```
// login in mini program
wx.login({
  success: res => {
    // this is your login code you need to send to casdoor
    console.log(res.code)

    wx.request({
      url: `${CASDOOR_HOSTNAME}/api/login/oauth/access_token`,
      method: "POST",
      data: {
        "tag": "wechat_miniprogram", // required
        "client_id": "6825f4f0af45554c8952",
        "code": res.code,
        "username": this.data.userInfo.nickName, // update user
profile, when you login.
        "avatar": this.data.userInfo.avatarUrl,
      },
      header:{
        "content-type": "application/x-www-form-urlencoded",
      },
      success: res => {
        console.log(res)
        this.globalData.accessToken = res.data.access_token // get
casdoor's accessToken
      }
    })
```

It is worth mentioning that the `tag` parameter is mandatory and you need to make casdoor understand that this is a request from the WeChat Mini Program.

The above code passes in the username and avatar uri of the WeChat Mini Program user while logging in. You can also pass these two parameters without passing them first, and then pass them to casdoor after the login is successful and accessToken is obtained:

```
wx.getUserProfile({
  desc: 'share your info to casdoor',
  success: (res) => {
    this.setData({
      userInfo: res.userInfo,
      hasUserInfo: true
    })
    console.log(app.globalData.accessToken)
    wx.request({
      url: `${CASDOOR_HOSTNAME}/api/update-user`, // casdoor uri
      method: "POST",
      data: {
        "owner": "test",
        "name": "wechat-oGk3T5tIiMFo3SazCO75f0HEiE7Q",
        "displayName": this.data.userInfo.nickName,
        "avatar": this.data.userInfo.avatarUrl
      },
      header: {
        "Authorization": "Bearer " + app.globalData.accessToken,
// Bearer token
        "content-type": "application/json"
      },
      success: (res) => {
        console.log(res)
      }
    })
  }
})
```

Also, you can use accessToken as a bearer token for any Casdoor operation you want.

> TIPS
>
> Currently Casdoor is unable to bind existing accounts to the WeChat Mini Program users. After Casdoor gets the openID from WeChat if this id does not exist, a new user will be created, and if it exists, the old one will be used.

# Lua

## APISIX

Using Casdoor in APISIX

# APISIX

Currently there are 2 methods to use Casdoor to connect to APISIX via APISIX plugins and protect the apis behind the APISIX: using APISIX's Casdoor plugin or using APISIX's OIDC plugin.

## Connect Casdoor via APISIX's Casdoor plugin

This plugin, authz-casdoor, can protect apis behind APISIX, forcing every single request to get authenticated, without modifying codes of api.

### How to enable it

You need to specify this plugin when creating the route, and give out all required fields. Here is an example.

```
curl "http://127.0.0.1:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "methods": ["GET"],
  "uri": "/anything/*",
  "plugins": {
    "authz-casdoor": {
        "endpoint_addr":"http://localhost:8000",
        "callback_url":"http://localhost:9080/anything/callback",
        "client_id":"7ceb9b7fda4a9061ec1c",
        "client_secret":"3416238e1edf915eac08b8fe345b2b95cdba7e04"
    }
```

In this example, using apisix's admin API we created a route "/anything/*" pointed to "httpbin.org:80", and with "authz-casdoor" enabled. This route is now under authentication protection of Casdoor.

## Attributes

| Name | Type | Requirement | Default | Valid | Description |
|---|---|---|---|---|---|
| endpoint_addr | string | required | | | The url of casdoor. |
| client_id | string | required | | | The client id in casdoor. |
| client_secret | string | required | | | The client secret in casdoor. |
| callback_url | string | required | | | The callback url which is used to receive state and code. |

*endpoint_addr and callback_url should not end with '/'*

In the configuration of "authz-casdoor" plugin we can see four parameters.

The first one is "callback_url". This is exactly the callback url in OAuth2. It should be emphasized that this callback url **must belong to the "uri" you specified for the route**, for example, in this example, http://localhost:9080/anything/callback obviously belongs to "/anything/*". Only by this way can the visit toward callback_url can be intercepted and utilized by the plugin(so that the plugin can

get the code and state in Oauth2). The logic of callback_url is implemented completely by the plugin so that there is no need to modify the server to implement this callback.

The second parameter "endpoint_addr" is obviously the url of Casdoor. The third and fourth parameters are "client_id" and "client_secret", which you can acquire from Casdoor when you register an app.

## How it works?

Suppose a new user who has never visited this route before is going to visit it (http://localhost:9080/anything/d?param1=foo¶m2=bar), considering that "authz-casdoor" is enabled, this visit would be processed by "authz-casdoor" plugin first. After checking the session and confirming that this user hasn't been authenticated, the visit will be intercepted. With the original url user wants to visit kept, he will be redirected to the login page of Casdoor.

After successfully logging in with username and password(or whatever method he uses), Casdoor will redirect this user to the "callback_url" with GET parameters "code" and "state" specified. Because the "callback_url" is known by the plugin, when the visit toward the "callback_url" is intercepted this time, the logic of "Authorization code Grant Flow" in Oauth2 will be triggered, which means this plugin will request the access token to confirm whether this user is really logged in. After this confirmation, this plugin will redirect this user to the original url user wants to visit, which was kept by us previously. The logged-in status will also be kept in the session.

Next time this user want to visit url behind this route (for example, http://localhost:9080/anything/d), after discovering that this user has been authenticated previously, this plugin won't redirect this user anymore so that this user can visit whatever he wants under this route without being interfered.

# Connect Casdoor via APISIX's OIDC plugin

Casdoor can use the OIDC protocol to link to APISIX, and this document will show you how to do it.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`APISIX_HOSTNAME`: Domain name or IP where APISIX is deployed.

## Step1. Deploy Casdoor and APISIX

Firstly, the Casdoor and APISIX should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. Set Casdoor's `origin` value (conf/app.conf) to `CASDOOR_HOSTNAME`.

```
conf > app.conf
dbName = casdoor
redisEndpoint =
defaultStorageProvider =
isCloudIntranet = false
authState = "casdoor"
httpProxy = "127.0.0.1:10808"
verificationCodeTimeout = 10
initScore = 2000
logPostOnly = true
origin = "http://10.144.1.2:8000"
                CASDOOR_HOSTNAME
```

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://APISIX_HOSTNAME/REDIRECTWHATYOUWANT`,and change `REDIRECTWHATYOUWANT` to the redirect url you need.
3. Select "JWT-Empty" for the Token format option
4. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, and we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

## Step3. Configure APISIX

APISIX has official OIDC support, which is implemented using lua-resty-openidc.

You can customize the settings according to the APISIX OIDC documentation, in which the following routing settings will be used:

```
#Use your own X-Api-Key
$ curl  -XPOST APISIX_HOSTNAME/apisix/admin/routes -H "X-Api-Key:
```

Now, visit `http://APISIX_HOSTNAME/get`, the browser will redirect you to the casdoor login page, and after successfully logging in, you will, not surprisingly, see that we have sent a request to httpbin.org.


```
{
  "args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Cookie": "casdoor_session_id=db6eed5e89bb70c19046856b5b719803; session=MUSCZ-upGgLbbDJbHEL2Iw..|1639973945|kTiWhzu8OmE3QBYbokh3WmPN6uJlxLeXe6dnl5QbpjWgvMq_veKDd0FY-
EWV0d1FQwDWa6uweDC4bh7M1mrKgABZ11P60dZ9HJebMhhmLW3dptnO38F3GhG9hDi7KoSD1OC0a4y7Koq1jNCodLL5zIJZ1K0usRQ3biWo54d8fXbZ_dmF05LrFFsxgYgp2-NegmaAHnE21HSkrD4wuXOZ6Q-ckfhIvEbU0C
4rxyugiSDthWSP1jfpLuWEb019gHNbKJPyKJ4YZ5EYvUIbepqrqyyTug5zF6WyIUTqY7TQ7aDQd8iUvetzcrswuj_ESNKTQtDVitaL9GkVqft0iswm2b0ryW9P_lB76ZIOtFtnnUR4_a6TzQjHaKYG9vtZRGSDIY1bTdHu0_F
y0Jlayp7sJltZO9bqurh2YXQGXoXWMxOPdZ44AQ15detkUycFeGAxllaFgfBQHQiyeCsE4XOJuxHv3gTKu9lYncwuhHJvzohI6KeZkK5P5xJhoZCfd3_yZb8a_ewU0dWXNruPqrsfd0yhYEPVHB-
3XEHdllJbW27tZEDXDzBEVmEAokS5QPDKZkIPwRtqqb6G1OKq4tXQWeNHPOrKaiItnB1EgD1rJbMKpV74nhYeihfsQnsamc_ZS6AaEFOMwAu1fROs1AsfCUj6kX47Q1_HV2VoKVfa2WAjPNhzrUAmrrzpwZBmn_lsFSa664f8
9d8U3jmU8wGhL_bRt8-AdRQOGYQht8ghJcat1OBIn6EUPB7G1CVsIaYYIBA2ylRBx6NV-J3NMRUsj5YkMbUqPFOLuR8JkhyYPs2spKMgcjGpAX6oaxVmtwc3U8yx5e4HHM3O2Q81MP2vW4lxffA81rqXWI4nOqClB-fQRJSEv
iCi6F0iFKvYMog7cfd4_LK9SIK6RTasyoJpPSEG26onqWSx8-jsRwNRykSb7eoYFThtAlOOXmkifgiJT53Id4JHP803RXsISguKPWtMjR6T1whIWQO0jr2wn182Xxqh8OprnlL8VLbLv_4e0p6uAbv4RnXfdE2P8Us6U0nNWz
oB6hrD99aM6NK_mxUZdoDGDlaRoeztXgNhv1yMO6SpKaer0xkbw7Gf3hNHq8Oh8xdkhx5QirUK24i2SoD9hq1f10pTG3ZjJffNY958EJMLxmfbsAcYN3C8or2WrMoVndKWe-LsAlrFydrkHd5cT7AUqc_znij8CUgsYj1C5d-
V1H8OgnbAu1Qvg6gym7qzBe6CcYn7B69CYT8wdrtwyHnGS7rurQy3cu9blq8J1WN9Nwf2Qgaovnoo LcHEUPVO1MBkqtM1Y8OCK7Ht-drf3RB6fNgOWaswbtPA4oUAX0Q9VHALhDzb9I7vBWB03UTqDBk37Oyv JWeOq6YfiY7z
jz4rjx8T8nM1OWEUUzQLNIVCH4bDt1GFX4Ga1A8hZT5W4WM0dcHL59XE4fXcHhtG18QGxCVrJ71ZzOsxTlugc1UEjzXGGsGX7DyODkRMt3n09AJwtKd8rqUATrc21ohgGJjLNWj4K_BOLMfW-ifJQKNxI6NcrbLGU-
P_nIvIiPLRNOdmCjySpShqdNPQCi3B9YiaBLLB04jBjzLNXkmkswf5LGFQEt16ZsTkcAbWuc1DgqjgFFESRdTS42G1wsUW97x0Q4QATA1WHR4Pyp3oR3tmpOG9GAcjnghc2POatcSOW5Ab70TVTKOVNYbLHnn6NT4tqgyeUDw
Isq4inM4UsmUG4AVzQvaXtPtYrEc03cOsfvFpoTOXH5ird92tLTZ36944oPg1Ou0r6OWxGbbs3Fi7P1_BLuo6WkkI5U_FgUrPjEDqrMs_BjhxpDcb4tKavfopUlc4_PhnOyk1pwCNki_UpFX-MyR6Sro-LyAOHN907eNXAPig
wxLHbQxklzT9qN_cp1DWmRFM9bEGuj4Vyn-1ZJNbSc49tGpLciaY-FfUXPn76At_mAfkJB1DAPT3ruhrs97T-W6k0mMxnWRz7BH4dQe0zYuDPcdqtqPcWZqiOeYjbhF1Jw9sXvj_w0kXBMnI7RNhKfU8a_8ZtBWpqvU9yYqP3
VEzho0kykCHcc_rmcfirndKnTcBDnMUbvcsikMFJpQFccEwZbB3Y73neTEYchs-Xp0ffSIhAq9BKtK-qWQCou2tYmCIMwvjx3X0XyB-CDyKpEX3oaxYuUzjgK2IR2FXRYhvFA1y4PAii3nvLNqT18f3oLrAZgago6To0i5Vv2
ka1DY5U2MN8NnKRTbaF6yOHgh3HWjdnI1J9GfhhaX9CFF2WOh0KZ0aLSHfyPHW5nI3AvcXyPlea1Tj_c1OK4qmo4TDPrdCiG3qjeOLS71gP6itjJBhdhjQe-PaQ855mcU4i3N5EuaQZSorJ3q21PiXENVqpaKBB1sZj5ansC-
sEh0r9q8s2iZfDk99DJGAzZU6GGCPGfJTOjFbbn2kI1Lox_sDg_1KUXMJv3h2pvQYIzS3nJFVvqj_-SGexIAM9ROzUuQU3F9IF1b_1aenlD0eHEnrQPxxsJMaIYV1_ACPLP_xvK4rMX7pY_pkl-oSQVjYu7v_LATxjQOW31gI
Axgrpiqn4b7AFIvXb4C9n7gQSvJqZwtnaU39VxBkGbWItP8HW4jxhaCS0GuYJTfUSc7AL004i-1N0EATyMv-S9Q0eYXwF5aNa1TVRyXtILFk5Gx8TQjwxMXiPeHPaz_NWpxra7GK1mag2GnAhPsPXAiG6GsKa5zZR5tAVc_MH
session_2=ItpAMrr7pWWwIAuQLPd1byNDNP5NhUVIDVopWYqkFWv3VPMz3VF0Kj01M64Vgd08SePmNV184wKtxmpZyrUXy1DwmG06UPv5Y9WwcKs8JvtQCVoseIdmSdDNMEfb1Jd62Q4vsqtkEJ_xxYW8mySYsMo2Gno_9eC
QptDzh1KzNCIMxHPVxpJFTZqbRyBkned6sKVRhN0df0O_bnmZla7aQYHhvaLfUXB9biZXbNkLbLr45J51RmprtTS0x8stKs1ZIWpM0kGvARYoiZQPahOs874Qv7V3F7pvxBwAJIr-Ov4LxIdnbc8ISi4qsvP4-dUhp46R76Ev
QNBAyoaRxHivZMASXvbpAjuLx3eUSZc0WBu81E5mq1H_CmlzhgDjBO8KPR61L7F1fUs5L3j4ZqF2JYjNBf72xrcZoa4eyJWmm2W2pNdJsLHCJYoHGj_7pkWZ19hKzfD-DK6qwLf6uA6VA9vV_KFvPz0s5vYD2UUutwx58uQU4
o4LY7CzU_60MVM0FA8VgSEJgbIhrhAJk7gcQiD5YvdGpjFi9mooYsEqDVe9I4dRyGO1K6c0P1Glv9Nf90bcm5Tsi8mrjr-Bcfb_THBTgapPCHhTtMt1gIbXvLkT-mnRC5chhPblumxzsBLkI8563t2CRCsjVFB_EvAt6n3CNs
mtvOhc2hJ50Mr8azIanuCisjETqSTgKP9BqiQkx1ejKCpPm2Sjgj8MNiMbEGiLwi8NJG81jkPzNuGJ0NZwFZv3xay9i5ZUkARIS7iaz8seTr3alNWpKaS8ccnHyAGrm-bMdc_tcQO8AgOsA-
ePBfVK4EoBCG4vrvTM_bBgdnoxmV8iBbb5eq4mZ1F8f74usI7XX4elkyto9QAJ18AwOYhnce0DZtQwxDpd7Me2WFZlmdi8ly_FjxXkyTrrSER7vQFBeDFNtbSfuiM6kiRYP53QSblJdiBaecPQJD3cjeCmAHp8cnP-VwZOvIF
ppf31IzSQOfU48scJWongdmaJ_ZjsUb_PUsxFU7TDxbIi2bCd3COgZR5JiaLESXyIT|ZBOG5HiNCt39_pCbipXoxnlwxy8.",
    "Host": "[redacted]",
    "Referer": "http://[redacted]0/",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36",
    "X-Access-Token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW11IjoiYWRtaW4iLCJvd251ciI6ImJ1aWx0LW1uIiwibm9uY2UiOiJmM2U5MzU2YTFmN2MwMjVmYzNhYmE0OTc4OWI0YjNiOSIsImlzcyI6Imh0dHA6Ly8xNjguMTV
V4cCI6MTY0MDU3NTE0NSwibmJmIjoxNjM50TcwMzQ1LCJpYXQiOjE2Mzk5NzAzNDV9.BEIAb5s8ywTRYAVDKgLn4thIQpDqlPBD83avwNcE214J-cjirIdxYeagOu6869HK_xUCmdHvfhEeZalhmfWbVwK3YUfMNTIodq7sr8
nvgGcJx393DSPnjJ5YcTvh2fdjB1c7YWGFVWFOypSwd61nOXxRlAfMJqKfKNfyEZMFzni_9VyStxY2C5oJk_0etky9vfuA2rTvNXAfMtAd1bQ_TvRHUD70ITkyBRKCddRzPg_IlK95zumwHqU_okjHvOBAs2SkTOq0wv7S_Xf
MhiHucQF1GR0aE2jIU2GLTJMMnOyHYfNKUshCuWn6BmiD6DeDqPSC-VUPcWRgZsS8pA0acTBbzatJmM7T6gAyA5i5MjhGmslpuRJc7j0539HbMMrYeiTv-RwOAqv_kE1x_RUKmvRoUhKWwQfzzaFnzExMSzlrR91Kn9txteE3
    "X-Amzn-Trace-Id": "Root=1-61bff62a-597370f76f7f7eda061ac1db",
    "X-Forwarded-Host": "[redacted]",
    "X-Userinfo":
"eyJtc2ciOiIiLCJzdWIiOiIOODNiZTBlYi02NjJiLTQ5YzMtODczZS03MTU0NjY0ZmQzNGEiLCJkYXRhIjp7Imxhcmsi0iIiLCJnaXRsYWIiOiIiLCJhcHBsZSI6IiIsImF6dXJlYWQiOiIiLCJzbGFjayI6IiIsInBhc3N3
sInRhZyI6InN0YWZmIiwicGFzc3dvcmQiOiIxMjMiLCJhZGRyZXNzIjp7fSwicHJvcGVydGllcyI6e3OsImxkYXAiOiIiLCJkaXNwbGF5TmFtZSI6IkFkbWluIiwiYXZhdGFyIjoiaHR0cHM6XC9cL2Nhc2Jpbi5vcmdcL2lt
lciI6ImJ1aWxOLWluIiwicmVnaW9uIjoiIiwibGFuZ3VhZ2Ui0iIiLCJzY29yZSI6MjAwMCwiY3J1YXR1ZFRpbWUiOiIyMDIxLTEyLTE0VDA1OjQ1OjE3WiIsImlzT25saW51IjpmYWxzZSwidXBkYXR1ZFRpbWUiOiIiLCJp
hbC1lc2VyIiwic2lnbnVwQXBwbGljYXRpb24iOiIiLCJoYXNoIjoiMjlmMDA5MzAzODNiOWMyM2U4MzAzNmI4OTg3YTlkNzAiLCJpZENhcmRUeXB1IjoiIiwiaWRDYXJkIjoiIiwiZWlhaWwiOiJhZG1pbkBleGFtcGxlLmNv
iOiIiLCJwcmVIYXNoIjoiMjlmMDA5MzAzODNiOWMyM2U4MzAzNmI4OTg3YTlkNzAiLCJjcmVhdGVkSXAiOiIiLCJsYXN0U2lnbmluVGltZSI6IiIsImxhc3RTaWduaW5JcCI6IiIsImdpdGh1YiI6IiIsImdvb2dsZSI6IiIs
iIn0sImRhdGEyIjp7InBob251UHJlZml4IjoiODYiLCJwYXNzd29yZFNhbHQiOiIiLCJkaXNwbGF5TmFtZSI6IkJ1aWx0LW1uIE9yZ2FuaXphdGlvbiIsImVuYWJsZVNvZnREZWxldGlvbiI6ZmFsc2UsImRlZmF1bHRBdmFC
odHRwczpcL1wvZXhhbXBsZS5jb20iLCJtYXN0ZXJQYXNzd29yZCI6IiIsImZhdmljb24iOiJodHRwczpcL1wvY2RuLmNhc2Jpbi5jb21cL3N0YXRpY1wvZmF2aWNvbi5pY28iLCJjcmVhdGVkVGltZSI6IjIwMjEtMTItMTRU
  },
  "origin": "119.[redacted].8",
  "url": "http:/[redacted]et"
}
```

# PHP

## Zentao

Using Casdoor for authentication in Zentao

## ShowDoc

Using Casdoor as oAuth2 server in ShowDoc

## Flarum

Using OAuth2 to connect various applications, like Flarum

## Moodle

Using Oauth to connect Moodle

# Zentao

Zentao is an agile(scrum) project management system/tool, but it does not support OIDC itself. For integrating Zentao with Casdoor SSO, we should via 3rd-party OIDC module zentao-oidc, and this document will show you how to do it.

## Step1. Deploy Casdoor and Zentao

Firstly, the Casdoor and Zentao should be deployed. After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used successfully.
2. You can successfully log in and use Zentao

## Step2. Integrated Zentao OIDC third party module

install zentao-oidc

```
git clone https://github.com/casdoor/zentao-oidc.git
```

or you can download the ZIP and unzip it.

This module is used for Zentao integrating with SSO for OpenId. The usage is as follows:

1. Copy the entire oidc directory to the Module of The Zentao and use it as a

module of the Zentao. Rename the downloaded package to "oidc"

2. Configure the filter

   Because the framework of Zentao filters the parameters in URL and does not allow Spaces. So you need to put the following code at the end of `/config/my.php`.

   ```
   $filter->oidc=new stdclass();
   $filter->oidc->index=new stdclass();
   $filter->oidc->index->paramValue['scope']='reg::any';
   ```

3. Modify `/module/commom/model.php`

   Put 'oidc' on the anonymous access list and add a line to the `isOpenMethod` method of `model.php`.

```
public function isOpenMethod($module, $method){
   if($module == 'oidc' and $method == 'index')  return true;
}
```

4. If you do not want the Zentao login screen to appear, go directly to the Casdoor login screen. Modify the last line of code at `public function checkPriv()` in `/module/common/model.php`.

```
//return print(js::locate(helper::createLink('user', 'login',
"referer=$referer")));
return print(js::locate(helper::createLink('oidc', 'index',
"referer=$referer")));
```

5. Modify `setSuperVars()` method inside of `framework/base/`

`router.class.php`, comment out the following statements.

```
    public function setSuperVars()
//  unset($_REQUEST);
```

# Step3. Configure Casdoor Application

1. Create or use an existing Casdoor application.
2. Add Your redirect url

3. Add provider you want and supplement other settings.

# Step4. Configure Zentao

Configure `config.php` in the oidc

```
$config->oidc->clientId="<Your ClientId>";
$config->oidc->clientSecret="<Your ClientSecrect>";
$config->oidc->issuer="http://localhost:8000";
```

set your reditrect Url in module/oidc `public function index()`

```
$oidc->setRedirectURL($path."/zentao/oidc-index.html");
```

> NOTE
>
> The URL here refers to calling the 'index' method in the 'oidc' module. You also need to set a variable separator, which the framework defaults to with a dash : -
> please refer to zentao's official framework for details. "zentaoPHP"

# ShowDoc

## Using Casdoor for authentication in ShowDoc

ShowDoc is an online API documentation, technical documentation tool perfect for IT teams. Showdoc makes it easy to use Markdown syntax to write beautiful API documents, data dictionary documents, technical documents, online Excel documents, and more.

Showdoc supports 3rd-party authentication including Oauth. Here is the tutorial for achieving this.

### step1. Create an Casdoor application

Go to your Casdoor and add your new application **Showdoc**. Here is an **example** of creating the Showdoc application in Casdoor.

Please remember the `client ID` and `client Secret` for next step.

> INFO
>
> Please don't fill in the **callback url** in this step. The url depends on the configurations on showdoc in next step. Later we will come back to set a correct callback url.

## step2. Configure Showdoc

First, start the oAuth2 login button. Then fill in the `callback url` as shown in the example. Fill in the `client ID` and `client secret` remembered in previous step.

`Authorize path`, `AccessToken path`, `User info path` are required. You can fill as shown below.

```
Authorize path:   /login/oauth/authorize
AccessToken path:   /api/login/oauth/access_token
User info path:   /api/get-account
```

## step3. Configure the callback url in casdoor

Go back to the application edit page in step 1, and add the `callback url` you filled in showdoc.

## step4. Have a try on showdoc

You are supposed to see this in login page:

Congratulations! You have completed all the steps. Press the **'casdoor sso'** button and you will be redirected to casdoor login page.

# Flarum

Casdoor can use OAuth2 to connect various applications. Here we will use Flarum as an example to show you how to use OAuth2 to connect to your applications.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Flarum_HOSTNAME`: Domain name or IP where Flarum is deployed.

## Step1. Deploy Casdoor and Flarum

Firstly, the Casdoor and Flarum should be deployed.

After a successful deployment, you need to ensure:

1. Download the Flarum plugin FoF Passport
2. Casdoor can be logged in and used normally.
3. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find a redirect url: `<CASDOOR_HOSTNAME>/auth/passport`
3. Add your redirect url to casdoor application:

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, we will use them in the next step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure Flarum

1. You should install a plugin FoF Passport
2. You should config this app

3. You can find Client Id and Client Secret in Casdoor application page.

- `Token server url`: http://**`CASDOOR_HOSTNAME`**/api/login/oauth/access_token
- `Authorization server url`: http://**`CASDOOR_HOSTNAME`**/login/oauth/authorize
- `UserInfo server url`: http://**`CASDOOR_HOSTNAME`**/api/get-account
- `Scopes`: `address phone openid profile offline_access email`

Log out of Flarum, and test SSO.

Log In
Cardoor SSO
Username or Email
Password
Remember Me
Log In
Forgot password?
Don't have an account? Sign Up

# Moodle

Casdoor can use Oauth to connect Moodle.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`Moodle_HOSTNAME`: Domain name or IP where Moodle is deployed.

## Step1. Deploy Casdoor and Moodle

Firstly, the Casdoor and Moodle should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find a redirect url: `Moddle_HOSTNAME`/admin/oauth2callback.php
3. Add your redirect url to casdoor application

More infomation for OAuth

# Step3. Configure Moodle

1. You should find OAuth

2. You should config this app

3. You should config this Mappling

**User field mappings for issuer: Casdoor**

| External field name | Internal field name | Edit |
|---|---|---|
| address | address | |
| email | email | |
| name | firstname | |
| phone | phone1 | |
| picture | picture | |
| perferred_username | username | |

Create new user field mapping for issuer "Casdoor"

4. Find OAuth2 plugin

General Users Courses Grades Plugins Appearance Server Reports Development

Your site is not yet registered. Register your site

**Plugins**

Install plugins
Plugins overview

**Activity modules**

Manage activities
Common activity settings
Assignment
Assignment settings
Submission plugins
Manage assignment submission plugins
File submissions
Online text submissions
Feedback plugins
Manage assignment feedback plugins
Feedback comments
Annotate PDF
File feedback
Offline grading worksheet
Book
Chat
Database
External tool
Manage tools
Feedback
File
Folder
Forum
Glossary
H5P
IMS content package
Lesson
Page
Quiz
General settings
Safe Exam Browser templates
Safe Exam Browser access rules
SCORM package
Text and media area
URL
Workshop

**Admin tools**

Manage admin tools
Accessibility
Brickfield registration
Accessibility toolkit settings
Reports
Recycle bin

**Antivirus plugins**

Manage antivirus plugins

**Authentication**

Manage authentication
Email-based self-registration
Manual accounts
OAuth 2

5. Enable OAuth2 plugin

6. if you want Casdoor's email can't be edited

More infomation for Moodle and Fields mapping

Log out of Moodle, and test SSO.

# Ruby

## GitLab

Using Casdoor for authentication in self-developed GitLab server

# GitLab

Casdoor can use the OIDC protocol to link to self-deployed GitLab server, and this document will show you how to do it.

> **CAUTION**
>
> As GitLab docs said, GitLab only works with OpenID providers that use HTTPS, so you need to deploy Casdoor with HTTPS, like putting Casdoor behind a NGINX reverse proxy with SSL certificate setup. Casdoor itself only listens to 8000 port by default via HTTP and has no HTTPS related functionality.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed. e.g., `https://door.casbin.com`.

`GITLAB_HOSTNAME`: Domain name or IP where GitLab is deployed. e.g., `https://gitlab.com`.

## Step1. Deploy Casdoor and GitLab

Firstly, the Casdoor and GitLab should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. Set Casdoor's `origin` value (conf/app.conf) to `CASDOOR_HOSTNAME`.

```
conf > app.conf
dbName = casdoor
redisEndpoint =
defaultStorageProvider =
isCloudIntranet = false
authState = "casdoor"
httpProxy = "127.0.0.1:10808"
verificationCodeTimeout = 10
initScore = 2000
logPostOnly = true
origin = "http://10.144.1.2:8000"
```

CASDOOR_HOSTNAME

# Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://GITLAB_HOSTNAME/users/auth/openid_connect/callback`.
3. Add provider you want and supplement other settings.

Not surprisingly, you can get two values on the application settings page: `Client ID` and `Client secret` like the picture above, and we will use them in the next

step.

Open your favorite browser and visit: **http://`CASDOOR_HOSTNAME`/.well-known/openid-configuration**, you will see the OIDC configure of Casdoor.

# Step3. Configure GitLab

You can follow the steps below to set this up, or make custom changes according to this document(e.g., you are installing GitLab using source code rather than Omnibus).

1. On your GitLab server, open the configuration file.

```
sudo editor /etc/gitlab/gitlab.rb
```

2. Add the provider configuration. (HOSTNAME url should include http or https)

```
gitlab_rails['omniauth_providers'] = [
    {
        name: "openid_connect",
        label: "Casdoor", # optional label for login button,
defaults to "Openid Connect"
        args: {
            name: "openid_connect",
            scope: ["openid", "profile", "email"],
            response_type: "code",
            issuer:  "<CASDOOR_HOSTNAME>",
            client_auth_method: "query",
            discovery: true,
            uid_field: "preferred_username",
            client_options: {
                identifier: "<YOUR CLIENT ID>",
                secret: "<YOUR CLIENT SECRET>",
```

3. Reboot your GitLab server.

4. Each registered user can open `GITLAB_HOSTNAME`/-/profile/account, connect the casdoor account.

5. Finish. Now, you can login your own GitLab by casdoor.

# Haskell

## Hasura

Before the integration, we need to deploy Casdoor locally.

# Hasura

Before the integration, we need to deploy Casdoor locally.

Then we can quickly implement a Casdoor-based login page in our own app with the following steps.

## Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Add a redirect url: `http://CASDOOR_HOSTNAME/login`

3. Copy the client ID, we will need it in the following steps.

## Add user in Casdoor

Now you have the application, but not a user. That means you need to create a user and assign the role.

Go to the “Users” page and click on “Add user” in the top right corner. That opens a new page where you can add the new user.

Save the user after adding a username and adding the organisation Hasura(other details are optional).

Now you need to set up a password for your user, which you can do by clicking manage your password.

Choose a password for your user and confirm it.

# Build Hasura App

Start the Hasura by docker or Hasura Cloud.

Now create a `users` table with the following columns:

- `id` of type Text (Primary Key)
- `username` of type Text

See the image below for reference.

The next step is to create a `user` role for the app. Users should be able to see only their records, but not the other people's records.

Configure the `user` role as shown in the image below. For more information, read about configuring permission rules in Hasura.

This way, users cannot read other people's records. They can only access theirs.

For testing purposes, add a dummy user. This is to ensure that when you use the JWT token, you only see your user's details and not other users' details.

Now you need to set the `JWT_SECRET` in Hasura.

# Configure Hasura with Casdoor

In this step, you need to add the **HASURA_GRAPHQL_JWT_SECRET** to Hasura.

To do so, go to the Hasura docker-compose.yaml and then add the new `HASURA_GRAPHQL_JWT_SECRET` as below.

The `HASURA_GRAPHQL_JWT_SECRET` should be in the following format:

```
HASURA_GRAPHQL_JWT_SECRET: '{"claims_map": {
      "x-hasura-allowed-roles": ["user","editor"],
      "x-hasura-default-role": "user",
      "x-hasura-user-id": "userID"
    },"jwk_url":"https://door.casdoor.com/.well-known/jwks"}'
```

Save the change, and reload the docker.

```
    ## enable debugging mode. It is recommended to disable this in production
    HASURA_GRAPHQL_DEV_MODE: "true"
    HASURA_GRAPHQL_ENABLED_LOG_TYPES: startup, http-log, webhook-log, websocket-log, query-log
    HASURA_GRAPHQL_ADMIN_SECRET: myadminsecretkey
    HASURA_GRAPHQL_JWT_SECRET: '{"claims_map": {
    "x-hasura-allowed-roles": ["user","editor"],
    "x-hasura-default-role": "user",
    "x-hasura-user-id": "4ec7ccee-ec7b-4191-a78d-e11f50686f8b"
  },"jwk_url":"https://door.casdoor.com/.well-known/jwks"}'
```

# Retrieve JWT Token

Since there is no client implementation, you can get your access token by making a request by below URL:

```
http://localhost:8000/login/oauth/authorize?client_id=<client
ID>>&response_type=code&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Flogin&scope=read&state=app-
built-in<public certificate>>
```

Change the client ID to the ID you copied before, and input the public certificate of Cadoor which you can find in Casdoor/Certs.

Then input the username and password you create for Hasura before.

Click Sign in.

Go back to Casdoor/Token page

Find the Username you input before then click edit

Copy the Access Token.

Now you can use the access token to make the authenticated request. Hasura returned the appropriate user rather than returning all the users from the database.

HASURA v2.22.0 | API | DATA | ACTIONS | REMOTE SCHEMAS | EVENTS | SETTINGS | HELP

GraphiQL | REST | Allow List

GraphQL Endpoint

Request Headers

| ENABLE | KEY | VALUE |
|---|---|---|
| ☐ | Hasura-Client-Name | casdoor |
| ☑ | content-type | application/json |
| ☐ | x-hasura-admin-secret | •••••••••••••••• |
| ☑ | Authorization | Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6ImNlcnQtYnVpbHQtaW4iLCJ0eXAiOiJKV1QifQ.eyJvd25lciI6Imhhc3VyYSIsIm5hbWL |
| | Enter Key | Enter Value |

Explorer

query MyQuery

- users
  - distinct_on:
  - limit:
  - offset:
  - order_by:
  - where:
  - id
  - username
- users_by_pk

GraphiQL | Prettify | History | Explorer | Code Exporter | REST | Derive action | Analyze | Docs

```
query MyQuery {
  users {
    id
    username
  }
}
```

QUERY VARIABLES

```
{
  "data": {
    "users": [
      {
        "id": "4ec7ccee-ec7b-4191-a78d-e11f50686f8b",
        "username": "test"
      }
    ]
  }
}
```

# Python

## JumpServer

Using CAS to connect JumpServer

# JumpServer

Casdoor can use CAS to connect JumpServer.

The following are some of the names in the configuration:

`CASDOOR_HOSTNAME`: Domain name or IP where Casdoor server is deployed.

`JumpServer_HOSTNAME`: Domain name or IP where JumpServer is deployed.

## Step1. Deploy Casdoor and JumpServer

Firstly, the Casdoor and JumpServer should be deployed.

After a successful deployment, you need to ensure:

1. Casdoor can be logged in and used normally.
2. You can set CASDOOR_HOSTNAME = `http://localhost:8000`. When deploy Casdoor in `prod` mode. See production mode.

## Step2. Configure Casdoor application

1. Create or use an existing Casdoor application.
2. Find a redirect url: `CASDOOR_HOSTNAME`/cas/`your orgnazition`/`your application`/login
3. Add your redirect url to casdoor application: `JumpServer_HOSTNAME`

More infomation for CAS

# Step3. Configure JumpServer

1. You should find Auth

2. You should config this app

- `/login` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/`

`login`

- `/logout` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/logout`
- `/serviceValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/serviceValidate`
- `/proxyValidate` endpoint: `https://door.casdoor.com/cas/casbin/cas-java-app/proxyValidate`

More infomation for CAS and JumpServer

Log out of JumpServer, and test SSO.

# Monitoring

## Web UI

Monitor runtime information on the casdoor Web page

## Prometheus

Use prometheus to collect information about running casdoor

# Web UI

You can monitor the runtime information of casdoor on the casdoor Web page, including CPU Usage, Memory Usage, API Latency, and API Throughput

You can view the following information on the UI

- CPU Usage and Memory Usage

- API Latency, Including count times and average latency

**API Latency**

| | | | |
|---|---|---|---|
| GET | /api/get-cert | 3 | 0.667 |
| GET | /api/get-certs | 27 | 1.333 |
| GET | /api/get-chats | 27 | 1.519 |
| GET | /api/get-default-application | 3 | 5.333 |
| GET | /api/get-email-and-phone | 1 | 1.000 |
| GET | /api/get-global-providers | 58 | 1.293 |

- API Throughput, Includeing total throughput and throughput per API

**API Throughput**

Total Throughput: 2

| Name | Method | Throughput |
|---|---|---|
| /api/get-prometheus-info | GET | 1 |
| /api/get-system-info | GET | 1 |

# Prometheus

Use Prometheus to collect casdoor's runtime metrics, including API Throughput, API Latency, CPU Usage, Memory Usage, and so on, as you should configure your Prometheus profile

```
global:
  scrape_interval: 10s #The time interval for fetching metrics

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'casdoor' #Name of the application to be monitored
    static_configs:
      - targets: ['localhost:8000'] #Back-end address of casdoor
deployment
    metrics_path: '/api/metrics' #Path for collecting indicators
```

After the configuration is successful, you will find the following information in Prometheus

# Internationalization

Casdoor supports multi-languages. By deploying the translations to Crowdin, we support Chinese, French, German, Russian, Japanese and Korean.

Casdoor uses the official Crowdin cli to sync translations from Crowdin. If you want to add more languages supports, please propose in our community, and if you want to help us speed up the translating work, please help us translate on Crowdin.

# Contributing to Casdoor

Welcome to Casdoor! This document is a guideline about how to contribute to Casdoor.

If you find something incorrect or missing, please leave comments / suggestions.

## Get involved

There are many ways to contribute to Casdoor. Here are some ideas to get started:

Use Casdoor and report issues! When using Casdoor, report issues to promote development of Casdoor, no matter bugs or proposal. Before filing an issue on GitHub, it would be better to discuss first on Discord or GitHub Discussions

> INFO
>
> When reporting an issue, please use English to describe the details of your problem.

Help with docs! Contributing start from docs is a good choice to start your contribution.

Help solve issues! We prepare a table containing easy tasks suitable for beginners, with different levels of challenges labeled with different tags, check the table here Casdoor Easy Tasks.

# Contributing

Now, if you are ready to create PR, here is the workflow for contributors:

1. Fork to your own
2. Clone fork to a local repository
3. Create a new branch and work on it
4. Keep your branch in sync
5. Commit your changes (make sure your commit message is concise)
6. Push your commits to your forked repository
7. Create a pull request from your branch to our **master** branch.

# Pull Requests

## Before you get started

Casdoor uses GitHub as its developing platform. So the pull requests are the main source of contributions.

There are something you need to know before you open a pull request:

- When you first pull request, you need to sign the **CLA**.
- Explain why you send this PR and what this PR would do to the repo.

- **One commit is allowed**. Make sure the PR does only one thing, otherwise please split it.
- If there are newly added files, please include Casdoor license to the top of new file(s).

```
// Copyright 2022 The Casdoor Authors. All Rights Reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
software
// distributed under the License is distributed on an "AS IS"
BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
// See the License for the specific language governing permissions
and
// limitations under the License.
```

## Semantic PRs

Your pull requests should follow the Conventional Commits spec. The basic requirement is that only the PR title or at least one commit message. For example, three commonly used PR titles are given below:

> ⚠ CAUTION
>
> The PR title must be in lower case.

1. **fix**: a commit of the *type* `fix` patches a bug in your codebase.

```
fix: prevent racing of requests
```

2. **feat**: a commit of the *type* `feat` introduces a new feature to the codebase.

```
feat: allow provided config object to extend other configs
```

3. **docs**: a commit of the *type* `docs` add or improve a document.

```
docs: correct spelling of CHANGELOG
```

For more details, please refer to Conventional Commits.

## Link PR with issue (if existed)

You can link a pull request to an issue to show that a fix is in progress and to automatically close the issue when the pull request is merged.

#### Linking a pull request to an issue using a keyword

You can link a pull request to an issue by using a supported keyword in the pull request's description or in a commit message. The pull request **must be** on the default branch.

- close
- fix
- resolve

Issue in the same repository, for example:

```
Fix: #902
```

For more details, please see Link PR to issue.

## Modify PRs

Inevitably, your PR may need to be revised. Please **re-use** the same PR when the code needs changes. Don't close the PR and open a new one.

Here is a possible example:

- Modify the code in your local.
- Modify this commit.

```
git commit --amend
```

- Push to your remote repository.

```
git push --force
```

Then the PR has been modified successfully! You can check it in Casdoor repository.

# Code Related

Some principles:

Readability - Important code should be well-documented. Code style should be complied with the existing one.

#### Naming convention

e.g., `signupUrl` for var names, `Signup URL` for UI

#### How to update i18n data?

Please note that we use Crowdin as translating platform and i18next as translating tool. When you add some words using i18next in the `web/` directory, you can run the `i18n/generate_test.go` to auto-generate the `web/src/locales/**/data.json`.

Run `i18n/generate_test.go`:

```
cd i18n && go test
```

All languages are filled in English by default. You are encouraged to help translate the newly added strings in the `web/src/locales/zh/data.json` by Crowdin after your PR has been merged.

> ⚠ CAUTION
>
> If you are not familiar with other language, please don't translate it. Keep the file as it is.

# License

By contributing to Casdoor, you agree that your contributions will be licensed under its Apache License.