# SPIP

Apprendre mettre en place un projet SPIP


- Initialisation du projet
- Premiers pas avec SPIP
- Plugins
- Syntaxe SPIP
- Aller plus loin

# Initialisation du projet

Tout d'abord commençons par mettre en place les premières brics du projet. Il sera intéressant d'utiliser *Docker Compose* pour faciliter l'installation, le déploiement et tout le reste.

Après avoir crée un répertoire pour votre projet, créez un fichier `docker-compose.yaml` a la racine en y ajoutant le contenu suivant:

```
services:
  db:
    image: mariadb:10.11
    restart: always
    container_name: spip_nom_projet_db
    volumes:
      - ./mariadb/conf.d:/etc/mysql/conf.d
      - ./mariadb/initdb.d:/docker-entrypoint-initdb.d
      - ./mariadb/data:/var/lib/mysql
    environment:
      - MYSQL_RANDOM_ROOT_PASSWORD=1
      - MYSQL_DATABASE=spip
      - MYSQL_USER=spip
      - MYSQL_PASSWORD=dockerdev

  app:
    image: ipeos/spip:4.4.6
    restart: always
    container_name: spip_nom_projet_app
    ports:
      - 8080:80
    links:
      - db:mysql
    environment:
      - SPIP_DB_SERVER=mysql
      - SPIP_DB_LOGIN=spip
      - SPIP_DB_PASS=dockerdev
      - SPIP_DB_NAME=spip
    volumes:
      - ./app/config:/var/www/html/config
```

```
- ./app/IMG:/var/www/html/IMG
- ./app/lib:/var/www/html/lib
- ./app/local:/var/www/html/local
- ./app/plugins:/var/www/html/plugins
- ./app/squelettes:/var/www/html/squelettes
- ./app/tmp:/var/www/html/tmp
```

Un fois ceci fait, vous pourrez a partir de votre terminal, demander a docker d'installer l'ensemble des paquets utile pour mettre en place votre serveur SPIP, en tapant ceci:

```
docker compose build
```

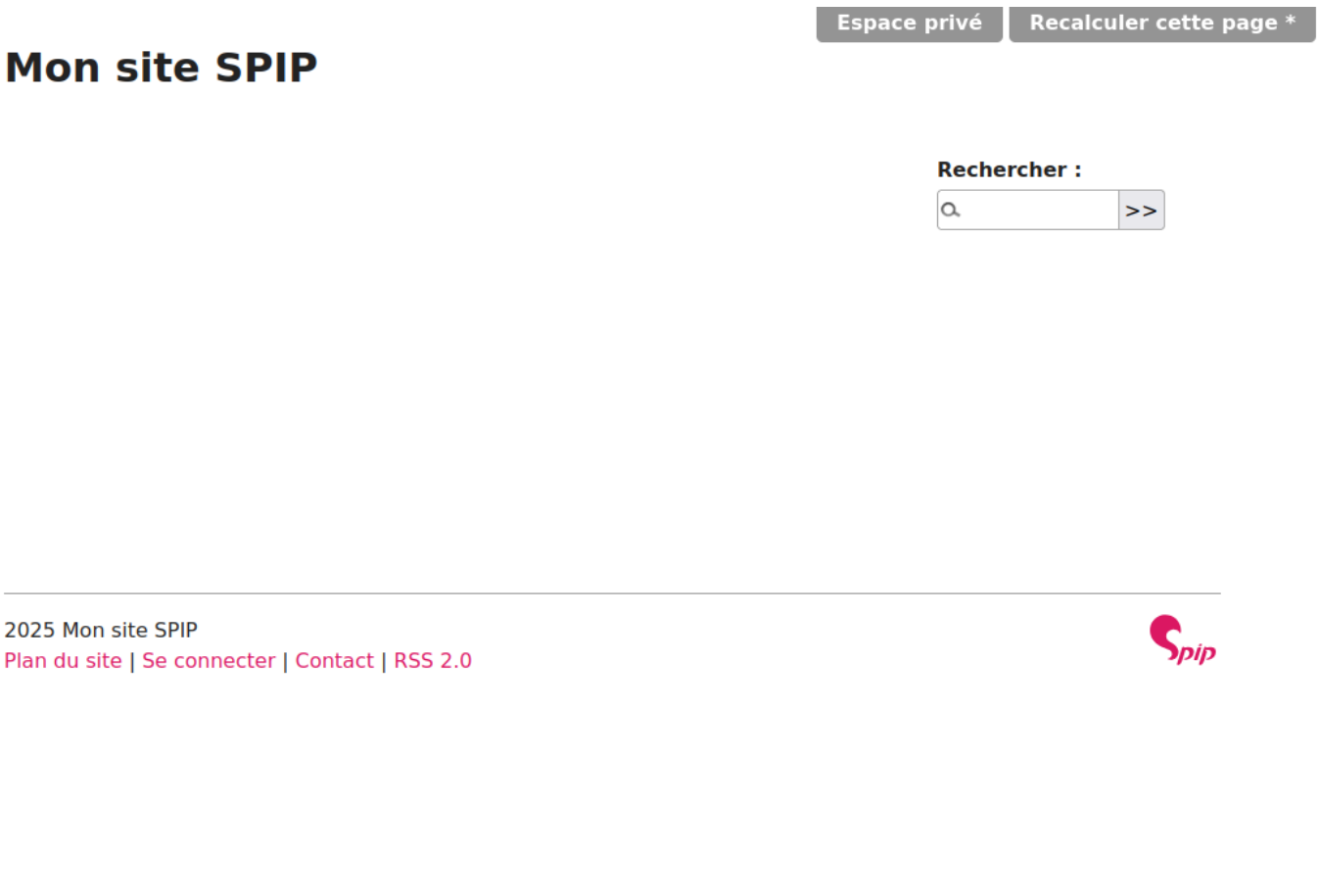
Maintenant que votre environnement de base est pret on peut lancer le serveur:

```
docker compose up -d
```

Si tout est bon, vous devriez pouvoir y accéder a l'adresse suivante : http://127.0.0.1:8880

> **NOTE:** Si vous voulez personnaliser la configuration de votre serveur, vous pourrez le faire directement dans votre fichier `docker-compose.yaml`. Vou pourrez retrouver l'ensemble des variables de l'image docker SPIP utilisé en cliquant ici.

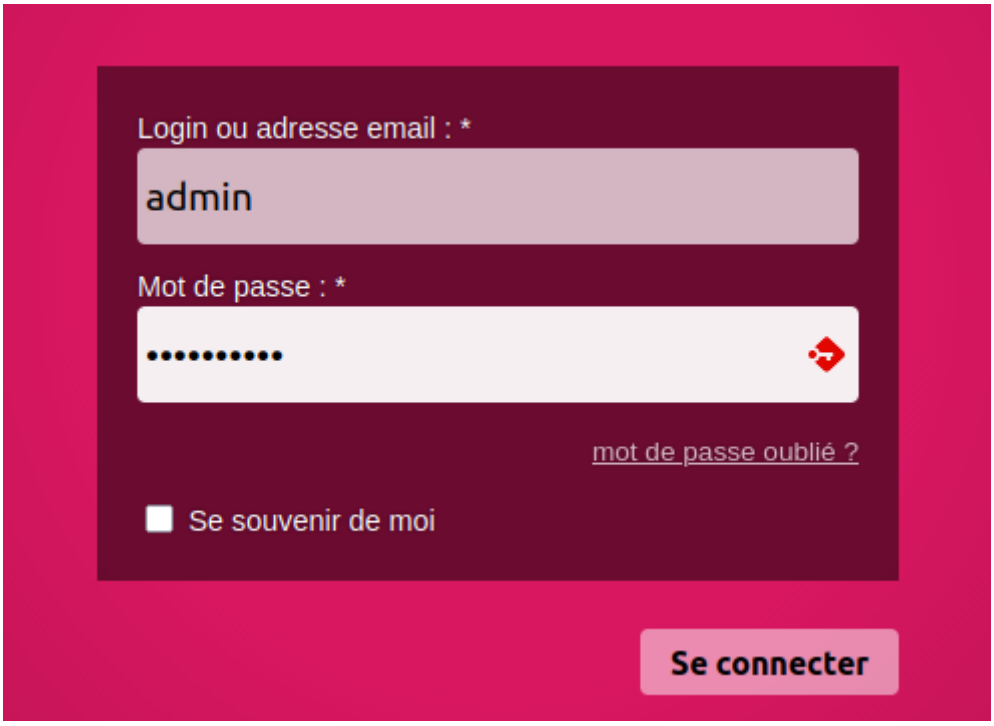
Sur l'interface web de votre serveur, vous devriez voir la page suivante:



Félicitations ! Vous avez votre site SPIP et vous etes sur la page d'accueil de votre site. Effectivement pour le moment elle est assez simpliste, mais ne vous inquiétez pas, très bientot nous verrons comment lui donner forme que vous voulez.

Nous allons avoir besoin d'accéder a l'interface réservé au administrateur. Vous pouvez cliquez sur le bouton "Espace privé" situé en haut.

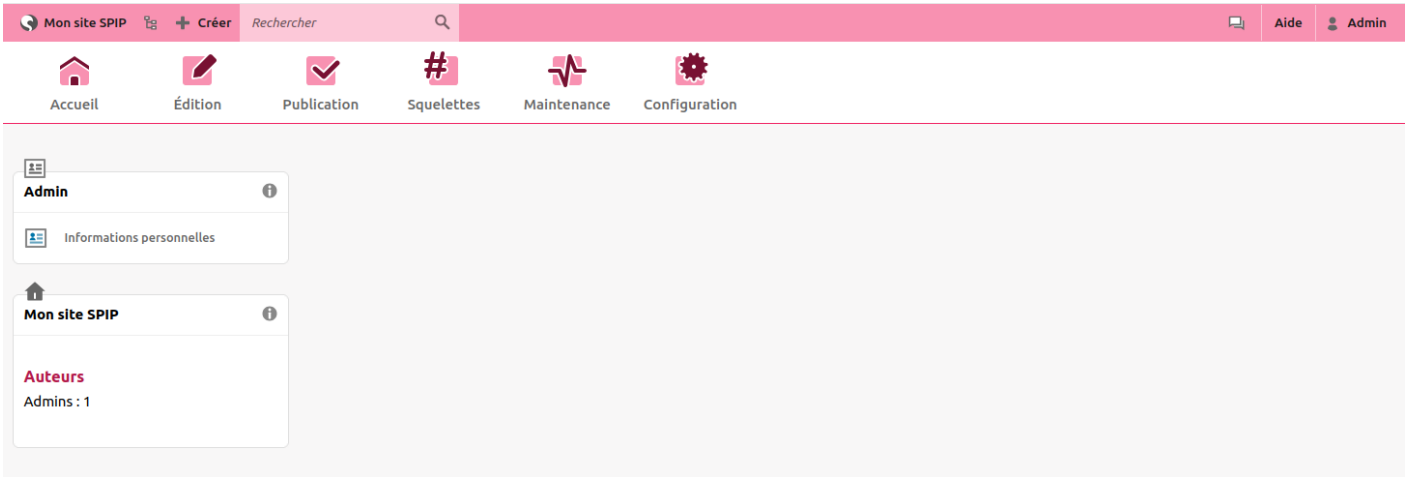
Vous allez arriver sur cette page, vous demandant de saisir vos identifiants:




Sauf si vous avez changez les variables `SPIP_ADMIN_LOGIN` et `SPIP_ADMIN_PASS`, ducoup dans ce cas il vous faudra saisir les valeur qui vous avez configuré, par défaut les identifants sont les suivant:

- **Login:** admin
- **Mot de passe:** adminadmin

Vous arrivez sur cette page:

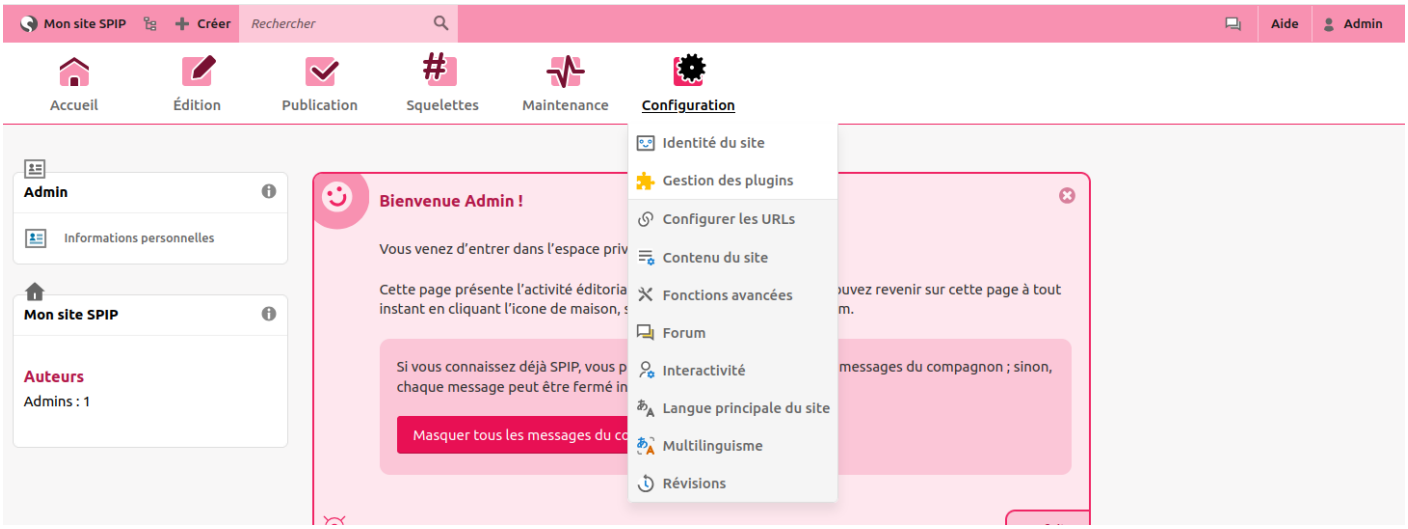


Bienvenue sur l'interface d'administration. En temps que Webmaster, c'est ici que vous passerez une grande partie de votre temps, afin de gréer le contenu de votre site, ajouter des plugins et autres.

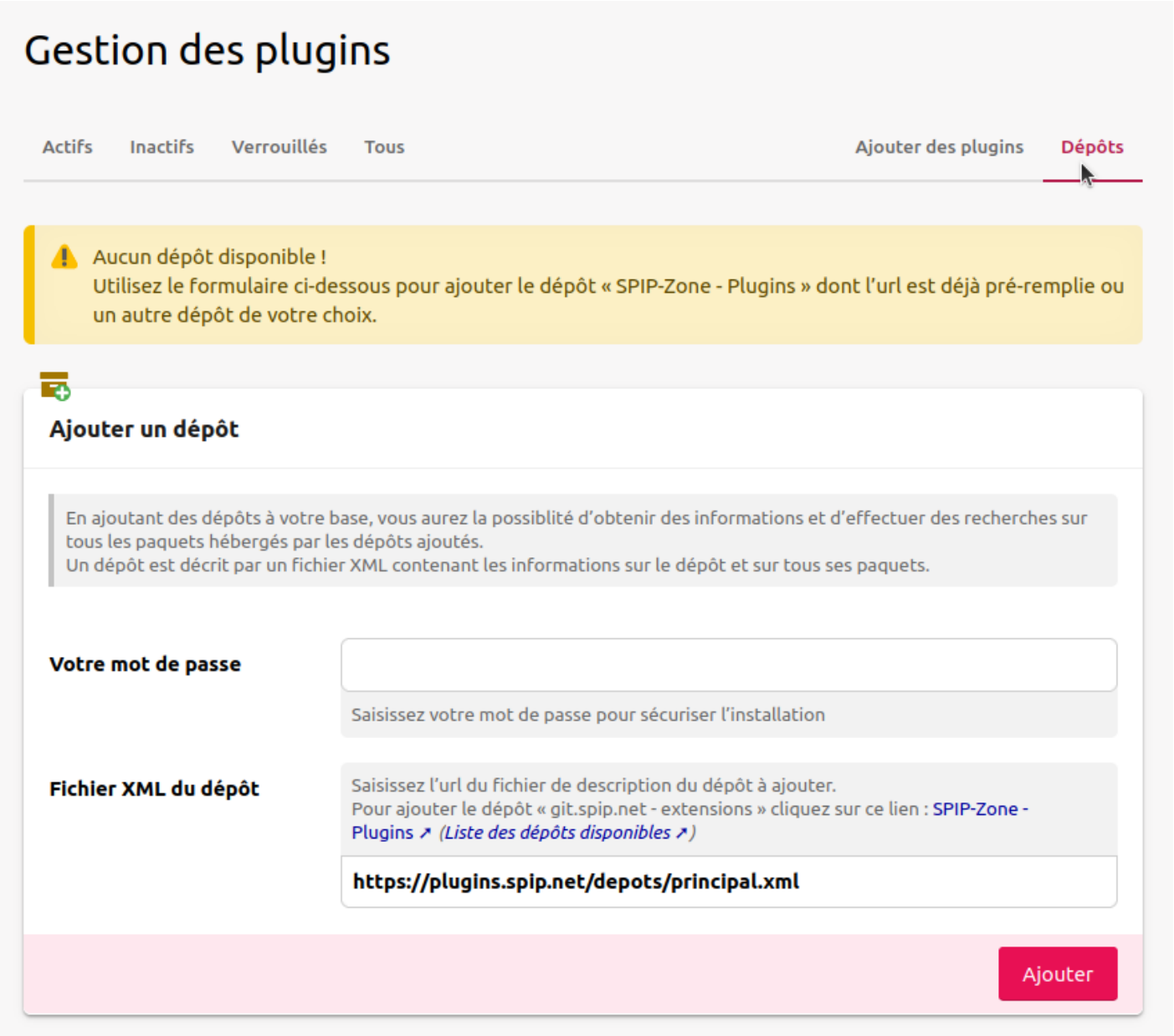
# Premiers pas avec SPIP

Maintenant que nous avons vu comment mettre en place notre serveur SPIP et comment accéder a l'interface d'administration. La première chose sera de rajouter les différentes dépots au projet afin de pouvoir faciliter l'installation des dépendances de nos plugins.

> **NOTE:** Un plugin SPIP est la jointure entre les scripts (PHP) et notre interface d'administration. Cela va par exemple nous etre utile notamment pour gérer notre site ou pour rajouter des comportement particulier, mais nous aurons le temps de voir ensemble des cas d'usages.

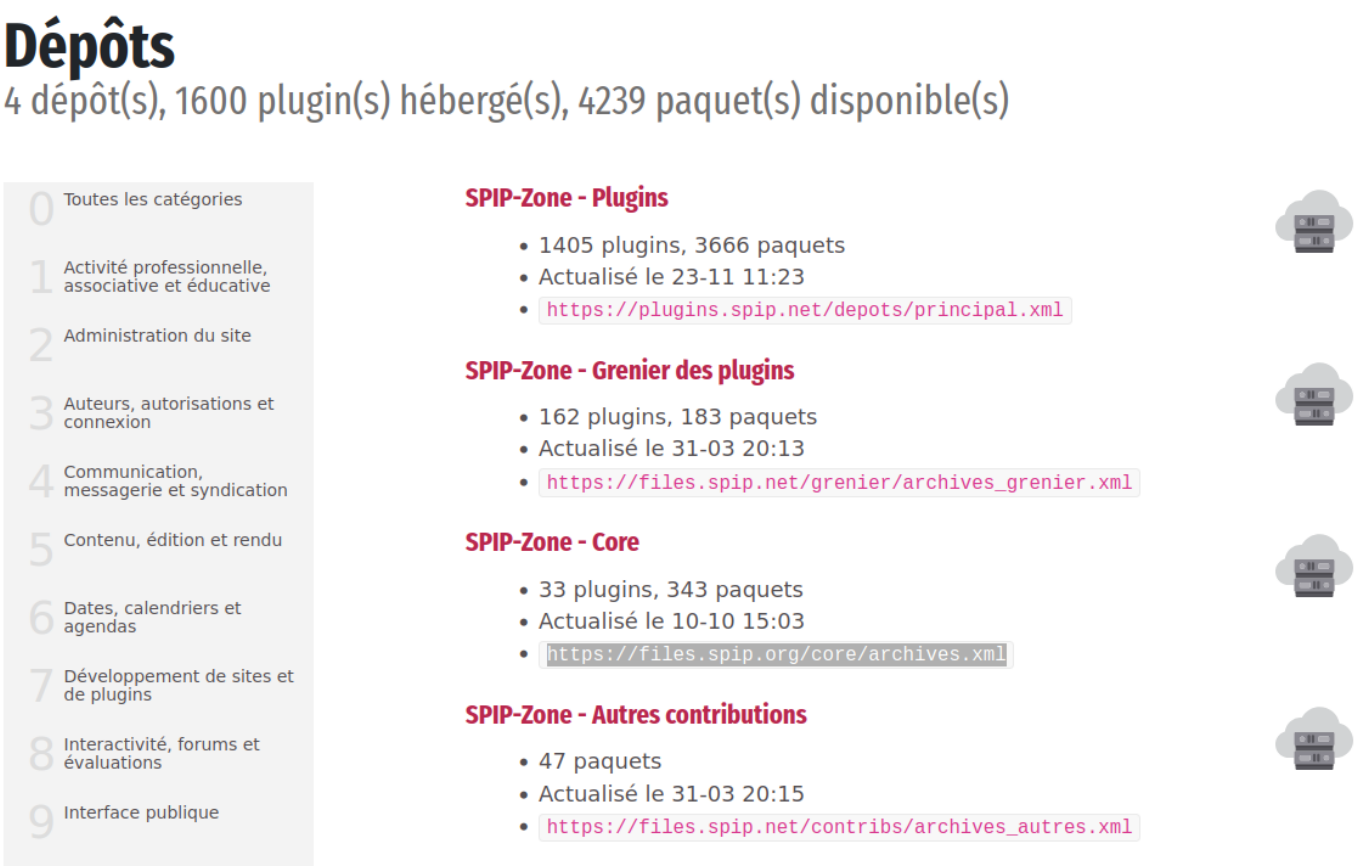



Pour cela, allez dans l'onglet "Configuration", puis "Gestion des plugins".




Cliquez sur "Dépôts" en haut a droite. Un premier dépôt est renseigné, vous n'aurez qu'a saisir le mot de passe (celui saisie pour vous connecté précédemment) et vous pouvez cliquer sur "Ajouter".

Il vous faudra certainement ajouter le dépôt SPIP-Core. Pour cela cliquez sur le lien "Liste des dépôts disponibles" situé au niveau de l'encadré a droite de "Fichier XML du dépôt". Accéssible également en cliquant ici.

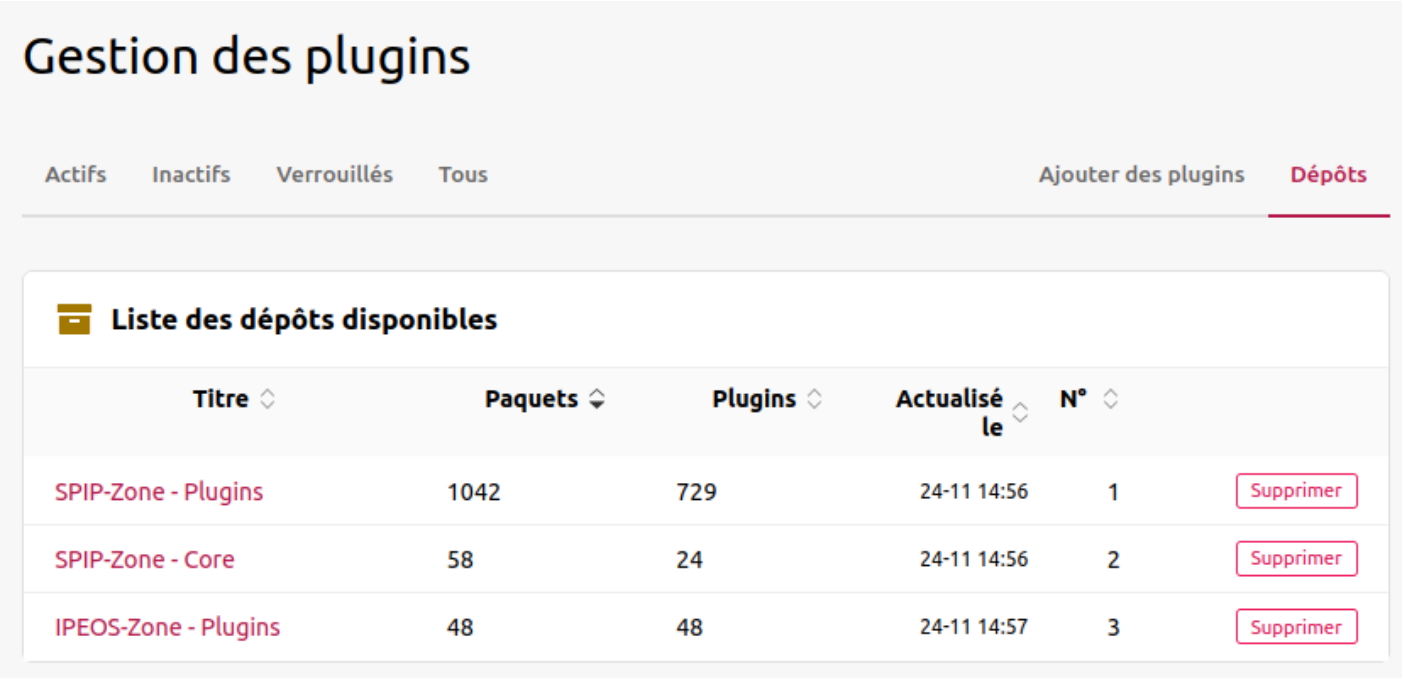



Sur cette page copiez l'URL de **SPIP-Zone - Core** (https://files.spip.org/core/archives.xml).

Retournez ssur l'écran vu précédemment et collez l'URL, avec votre mot de passe pour ajoutez également ce dépôt.

Ajoutez également ce dépôt : https://repo.apps-ipeos.com/spip/liste.xml.

Vous devriez donc voir l'ensemble des dépôts que nous venont d'ajouter au dessus:




.

Au niveau de votre répertoire, la ou se trouve votre projet avec le fichier `docker-compose.yaml`, vous devriez avoir une structure de dossiers semblable a:

```
votre-projet/
├─ app/
│  ├─ config/
│  ├─ IMG/
│  ├─ lib/
│  ├─ local/
│  ├─ plugins/
│  │  ├─ auto/
│  ├─ squelettes/
│  ├─ tmp/
├─ mariadb/
├─ plugins_data/
docker-compose.yaml
```

Il nous faudra configurer les droits d'accès pour la suite.

D'abord se mettre les droits sur le dossier `app/plugins`:

```
sudo chown -R <votre nom d'utilisateur>: app/plugins
```

> **IMPORTANT:** Remplacez par votre nom d'utilisateur sur votre machine

Ensuite ddonnez les droits d'accès a `www-data` sur le dossier `app/plugins/auto`:

```
sudo chown -R www-data app/plugins/auto
```

Bravo ! Votre projet est plutot bien préparez pour la suite.

Vous pouvez créer un plugin qui va vour servir a ochestrer vos templates et faciliter la maintenabilité de votre site.

```
...
plugins/
├─ auto/
├─ itemplate-<nom de votre projet>/
```

> **REMARQUE:** `itemplate-...` est le nommage généralement employé pour un plugin de gestion du template propre au site, mais afin de faciliter la suite, nous appelerons ce dossier `monplugin` afin que se oit plus générale aux différentes cas d'uasages que l'on peut avoir.

Nous allons ajouter dans ce dossier d'autres éléments, afin d'indiquer a SPIP qu'il s'agit bien d'un plugin. Ajoutez les éléments suivants, afin d'avoir la même arborescence:

```
...
monplugin/
├─ lang/
├─ paquet.xml
```

Nous verrons ensuite en détail comment fonctionne un plugin.

# Plugins

Voici à quoi ressemble la structure minimale d'un plugin:

```
monplugin/
├─ compositions/
│  ├─ article-pleine-page.html
│  ├─ article-pleine-page.xml
│  ├─ rubrique-actualites.xml
│  ├─ rubrique-actualites.html
├─ css/  # (surcharges css de ibootstrap)
│  ├─ plugins/
│  │  ├─ _owl_carousel.scss
│  ├─ config_variables.scss.html  # (fichier pour récupérer des variables SPIP avec la syntaxe SPIP)
│  ├─ fonts.css
│  ├─ main.scss
│  ├─ _*.scss  # (fichiers SCSS du thème à intégrer avec @import dans main.scss)
├─ fonts/
├─ formulaires/
│  ├─ configurer_monplugin.html
├─ images/
├─ inclure/
│  ├─ menu.html
│  ├─ share_social_links.html
│  ├─ social_links.html
├─ js/
│  ├─ widgets/
│  │  ├─ owl_carousel.js
│  ├─ main.js  # (indispensable)
├─ lang/  # (indispensable)
│  ├─ monplugin_fr.php
│  ├─ paquet-monplugin_fr.php.bak
├─ layouts/
│  ├─ inc/  # (surcharges de ibootstrap)
│  │  ├─ footer-defaut.html
│  │  ├─ header.html
│  │  ├─ navigation.html
```

```
│  │  ├─ page-header.html
│  │  ├─ page.html
├─ pages/  # (surcharges de pages existantes de ibootstrap)
│  ├─ sommaire/
│  │  ├─ inclure/
│  │  │  ├─ about.html
│  │  │  ├─ contact.html
│  │  │  ├─ header.html
│  │  │  ├─ news.html
│  │  │  ├─ services.html
│  │  ├─ content.html
│  │  ├─ aside.html
│  │  ├─ pre-content.html
├─ prive/
│  ├─ squelettes/
│  │  ├─ contenu/
│  │  │  ├─ configurer_ibootstrap.html  # (surcharge de ibootstrap)
│  ├─ themes/
│  │  ├─ spip/
│  │  │  ├─ images/
│  │  │  │  ├─ monplugin-16.png
│  │  │  │  ├─ monplugin-24.png
│  │  │  │  ├─ monplugin-32.png
│  │  │  │  ├─ monplugin.png
├─ vendor/
│  ├─ owl.carousel/
│  │  ├─ css/
│  │  │  ├─ owl.carousel.min.css
│  │  │  ├─ owl.theme.default.min.css
│  │  ├─ js/
│  │  │  ├─ owl.carousel.min.js
├─ paquet.xml  # (indispensable)
├─ monplugin_fonctions.php
├─ monplugin_pipelines.php
├─ favicon.ico
├─ imentionslegales.js.html  # (pour les cookies)
```

Mais comme nous l'avons vu précédemment, il n'y a que le fichier `paquet.xml` et le dossier `lang` qui sont toujours indispensable.

### Exemple de `paquet.xml`

```
<paquet
    prefix="monplugin"
    categorie="squelette"
    version="0.0.1"
    etat="dev"
    compatibilite="[4.0.0;["
>
    <nom>Mon plugin</nom>
    <auteur>Nom Prénom</auteur>
    <licence lien="http://www.gnu.org/licenses/gpl-3.0.html">GPL 3</licence>

    <!-- Pipelines utilisés -->
    <!-- Exemple insert_head implémenté dans le fichier monplugin_pipelines.php -->
    <pipeline nom="insert_head" inclure="monplugin_pipelines.php" />
    <pipeline nom="insert_head_css" inclure="monplugin_pipelines.php" />

    <!-- Dépendances du plugin -->
    <!-- Exemple plugin ibootstrap version 6.0.0 ou plus -->
    <necessite nom="ibootstrap" compatibilite="[6.0.0;[" />

</paquet>
```

Le `pipeline insert_head` permet d'ajouter des scripts dans le head des pages publiques du site.

# Les compositions

Les **compositions** permettent de proposer plusieurs mises en page différentes pour un même type d'objet (article, rubrique, etc.).

Pour chaque composition, vous devez créer **deux fichiers** dans le répertoire `compositions/` :

1. **Un fichier HTML** : le squelette de la composition
2. **Un fichier XML** : le descriptif de la composition

### Exemple : composition "Actualités" pour les rubriques

**Fichier `compositions/rubrique-actualites.html`** :

```
<BOUCLE_principale(RUBRIQUES) {id_rubrique}>
    <INCLURE{fond=layouts/#CONFIG{ibootstrap/layout,layout},env}{bs_page=rubrique-
actualites}{composition=#COMPOSITION} />
</BOUCLE_principale>
```

**Fichier `compositions/rubrique-actualites.xml` :**

```
<composition>
    <nom><:monplugin:rubrique_actualites:></nom>
    <description><:monplugin:rubrique_actualites_desc:></description>
    <icon>images/rubrique-actualites.png</icon>
</composition>
```

Une fois les fichiers créés, la composition apparaît dans l'interface d'édition de l'objet concerné (article, rubrique...).

Les rédacteurs peuvent alors choisir la composition souhaitée dans le menu déroulant "Composition".

# Surcharge des squelettes de pages

SPIP utilise un système de **hiérarchie de fichiers** pour déterminer quel squelette afficher.

Une surcharge sert à personnaliser une page existante (d'un plugin parent comme `ibootstrap`), créez un fichier avec le **même nom** dans votre plugin. SPIP utilisera automatiquement votre version.

## Hiérarchie de recherche des squelettes

SPIP cherche les squelettes dans cet ordre :

1. **Répertoire `squelettes/`** (à la racine de SPIP)
2. **Plugins activés** (ordre de priorité des plugins)
3. **Répertoire `dist/`** (squelettes par défaut de SPIP)

## Exemples de surcharges courantes

**Surcharge de la page d'accueil:**

Créez `sommaire.html` à la racine de votre plugin ou dans `squelettes/`, pour ajouter une variable d'env `pleine_page` :

```
<INCLURE{fond=layouts/#CONFIG{ibootstrap/layout,layout},env}{bs_page=sommaire}{breadcrumb=non}{pleine_page=oui}
/>
```

**Surcharge de la page auteur.html pour la désactiver:**

Créez `auteur.html` et laissez-le vide.

**Surcharge partielle avec `layouts/`:**

Pour surcharger uniquement des **fragments** de mise en page (header, footer, navigation...), utilisez le répertoire `layouts/inc/`. Par exemple dans le fichier `layouts/inc/header.html` (créez le sil n'existe pas).

Collez le code suivant:

```
<INCLURE{fond=layouts/inc/navigation,env} />
[(#ENV{bs_page}|=={sommaire}|non)<INCLURE{fond=layouts/header-objet, env} />]
```

Cette structure permet de surcharger seulement le header tout en conservant la structure principale du plugin parent.

## Ordre de priorité avec iBootstrap

Si vous utilisez le plugin `ibootstrap`, voici comment SPIP recherche les fichiers :

1. `monplugin/layouts/inc/header.html` (votre surcharge)
2. `ibootstrap/layouts/inc/header.html` (version du plugin)
3. `dist/header.html` (version par défaut)

# Pipelines

Les pipelines sont des points d’entrée prévus par SPIP qui permettent aux plugins d’intervenir à différents moments du fonctionnement du site. Ils offrent la possibilité d’ajouter, modifier ou filtrer du code et du contenu, que ce soit lors de l’affichage des pages, du traitement des formulaires ou d’autres étapes du cycle de vie de SPIP.

**Exemple de fichier `monplugin_pipelines.php`**

```
<?php


if (!defined('_ECRIRE_INC_VERSION')) {
```

```
    return;
}

function monplugin_insert_head($flux) {
    $flux .= "<script src='" . find_in_path('/js/widgets/owl-carousel.js') . "' type='text/javascript'
defer></script>\n";
    return $flux;
}
```

# Syntaxe SPIP

# Syntaxe SPIP

SPIP possède son propre système de templates : **boucles**, **critères**, **balises**, **filtres**. Ce langage est très lisible et orienté contenu.

## Les boucles

Les **boucles** permettent d'interroger les objets éditoriaux (articles, rubriques, auteurs...).

Syntaxe générale :

```
<BOUCLE_nom(TABLE){critères}>
  ... contenu affiché pour chaque résultat ...
</BOUCLE_nom>
```

Exemple : afficher les 5 derniers articles d'une rubrique :

```
<BOUCLE_articles(ARTICLES){id_rubrique}{par date}{inverse}{0,5}>
  <h2>#TITRE</h2>
  <p>#INTRODUCTION</p>
</BOUCLE_articles>
```

## Les critères de boucles

Ce sont les éléments entre `{}` dans une boucle. Ils servent à filtrer, classer ou limiter les résultats. Ils correspondent à des éléments de requête SQL (ex: {par date} => ORDER BY date)

Exemples utiles :

| Critère | Rôle |
|---|---|
| `{id_rubrique}` | Filtre par rubrique courante |
| `{par date}` | Tri par date ASC |
| `{par date}{inverse}` ou `{!par date}` | Tri par date DESC |
| `{0,10}` | Limiter aux 10 premiers résultats |
| `{doublons}` | Exclure des éléments déjà affichés |

# Les balises

Les **balises** correspondent généralement à une colonne d'une table si l'on est dans une boucle, elles peuvent aussi correspondre à des fonctions PHP déclarées sous la forme de `function balise_XXXX_dist()` généralement dans le fichier monplugin_fonctions.php.

Exemples :

```
#TITRE
#TEXTE
#DATE
#URL_ARTICLE
#LOGO_ARTICLE
```

Dans une boucle sur `ARTICLES`, `#TITRE` correspond au titre de l'article courant et `#LOGO_ARTICLE` exécute une fonction qui récupère le document identifié comme logo de cet article et renvoie une balise html `<img>`.

# Les filtres

Les **filtres** correspondent également à des fonctions PHP que l'on applique sur des balises avec ou sans argument.

Syntaxe : `#BALISE|filtre{argument}`

Exemples :

```
#TITRE|strtolower #TEXTE|couper{200} #DATE|affdate
```

# Aller plus loin

## Documentation essentielle SPIP

Voici les ressources officielles indispensables :

- **Guide Webmestres (structure des squelettes, fichiers, bonnes pratiques)** https://www.spip.net/fr_rubrique135.html
- **Boucles et balises** (référence complète) https://www.spip.net/fr_article894.html
- **Manuel des filtres** https://www.spip.net/fr_rubrique567.html
- **Programmer avec SPIP 4** (documentation technique générale) https://programmer.spip.net/
- **Tutoriels pédagogiques sur boucles / balises / critères** https://www.spippourlesnuls.fr/debuter/les-boucles/