# Installation

Le guide part du principe que tous les composants sont installés sur la même machine, mais gardez à l’esprit qu’il est tout à fait possible de les déployer sur plusieurs.

## Etapes obligatoires

Que ce soit une installation manuelle ou par Docker, il y a quelques prérequis communs.

### Installer GIT

TODO

### Récupérer les sources

TODO

### Installer le certificat

TODO

## Installation manuelle

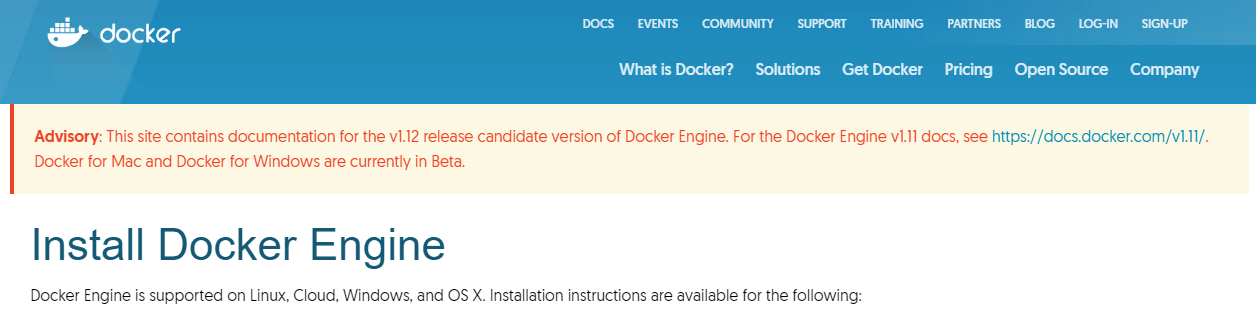
## Installer avec docker

### Prérequis

Le produit peut être déployé dans plusieurs conteneurs Docker. La procédure est assez simple à suivre et se résume en quelques étapes. Mais avant de commencer vous devez vous assurer que les prérequis suivants ont bien été installés et configurés sur votre machine.

#### Installer Docker

Docker doit être installé sur votre machine. Vous pouvez suivre le guide sur le site officiel ([lien](https://docs.docker.com/engine/installation/)) qui est très bien expliqué :



#### Configurer VirtualBox

Par défaut Docker utilise VirtualBox pour lancer sa machine virtuelle. Des règles de redirection de ports doivent être ajoutées sur la machine « default », sans quoi vous ne pourrez pas naviguer sur le site :

* Ouvrez VirtualBox et sélectionnez la machine « default »
* Cliquez sur « Configuration » puis sélectionnez l’onglet « network »
* Cliquez sur le bouton « port redirection » et assurez-vous de bien avoir les mêmes règles :

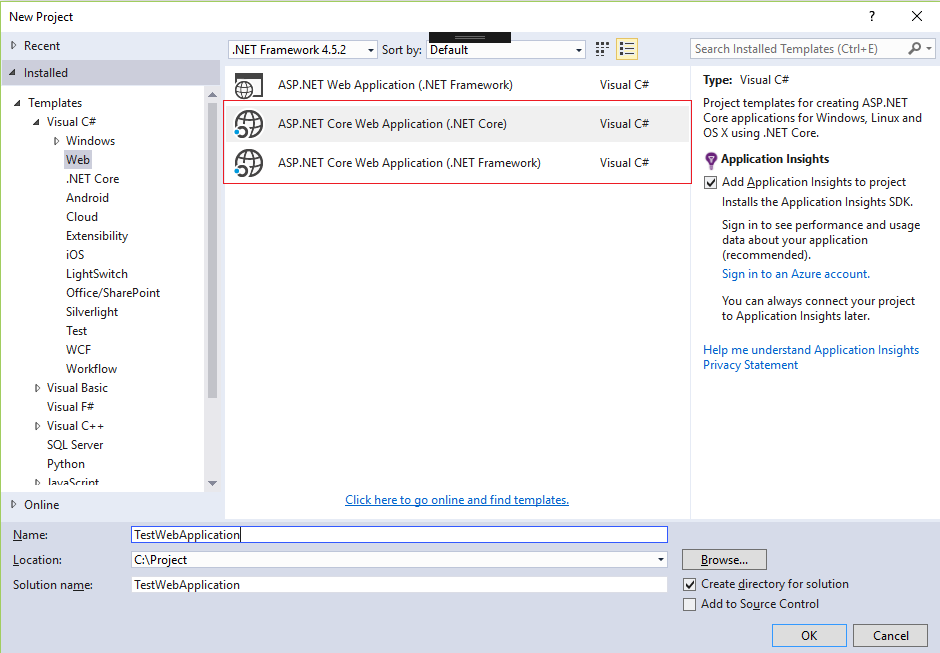


Mais avant de commencer, vous devez être sûr que ces

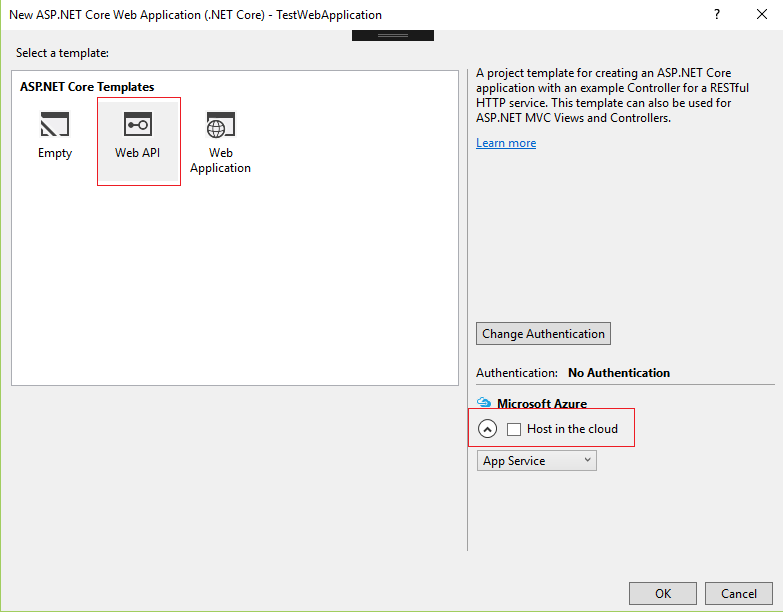
# Extension visual studio

## Projet WEB.API

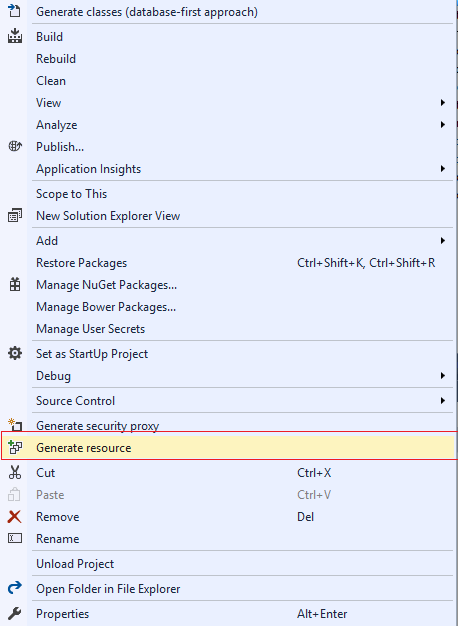
Créer une nouvelle solution avec un projet ASP.NET CORE WEB API. Pour cela dans la fenêtre « new project » sélectionnez le template « *ASP.NET Core Web Application (.NET CORE*) » ou « *ASP.NET Core Web Applications (.NET Framework)* ».  
Rentrez le nom du projet « TestWebApplication » puis cliquez sur « OK »



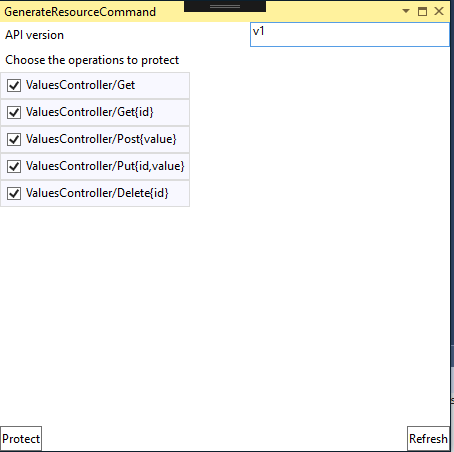
Dans la fenêtre suivante sélectionnez le type de projet « WEB.API », décochez « host in the cloud » et confirmer la création en cliquant sur « OK ».



Lorsque votre projet est créé et compilé, vous pouvez commencer à protéger les méthodes des contrôleurs. Pour cela sélectionnez votre projet « TestWebApplication », affichez le « context menu » et sélectionnez l’option « generate resource »



Une nouvelle fenêtre Visual Studio s’affiche. Attendez quelques secondes le temps que les opérations des contrôleurs soient listées. Si aucune opération n’est visible alors cliquez sur le bouton « refresh » et ré attendez quelques secondes.



Sélectionnez les opérations que vous souhaitez protéger, spécifier le numéro de version de votre API et cliquez sur « Protect ».

L’opération effectuera plusieurs tâches :

**Note** : S’il y a eu une erreur durant l’installation des packages, vérifiez que le feed a bien été ajouté. Deux Nuget packages doivent être installés :

🡺 SimpleIdentityServer.UmaIntrospection.Authentication (version 1.0.0)

🡺 SimpleIdentityServer.Uma.Authorization (version 1.0.0)

Si vous n’avez observé aucune erreur alors vous pouvez modifier votre fichier « Startup.cs » comme il est expliqué dans « Startup\_Sample.cs ».

Activer l’authentification en ajoutant dans la méthode « Startup.cs\Configure » le code suivant :

var options = new UmaIntrospectionOptions

{

ResourcesUrl = "http://localhost:8080/api/vs/resources",

UmaConfigurationUrl = "http://localhost:5001/.well-known/uma-configuration"

};

app.UseAuthenticationWithUmaIntrospection(options);

N’oubliez pas de vérifier et mettre à jour les valeurs des propriétés « ResourcesUrl » & « UmaConfigurationUrl » avec les bonnes URLs.

Vous pouvez choisir entre activer l’autorisation par convention ou par ressource.

La première se base sur une convention de nommage (voir Annexe 1) tandis que pour l’autre l’URL de la ressource doit être passée en paramètre.

services.AddAuthorization(options =>

{

options.AddPolicy("uma", policy => policy.AddConventionalUma());

});

services.AddAuthorization(options =>

{

options.AddPolicy("resourceSet", policy => policy.AddResourceUma("values/get", new List<string>

{

"execute"

}));

});

Afin de rendre effectif l’autorisation, les opérations ou contrôleurs doivent être décorées par l’attribut [Authorization] et le nom de la politique (resourceSet ou uma) doit être passé en paramètre.

[Authorize("resourceSet")]

public IEnumerable<string> Get()

{

return new string[] { "value1", "value2" };

}

# Annexe 1 : convention de nommage

**Le patron** : resources /Apis / <nom du projet> / <numéro de version> / <nom du contrôleur > / <nom de l’action> ({liste des paramètres})

# Getting Started

## Install

### Step-by-step

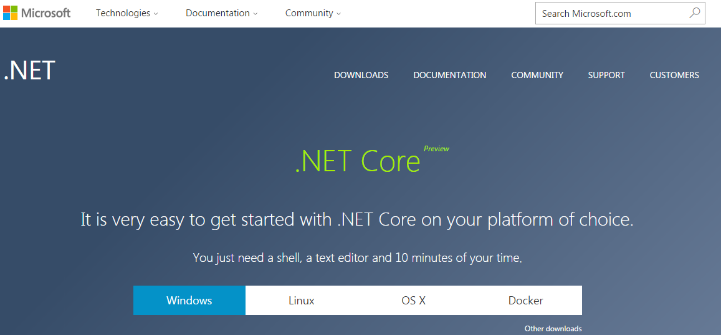
Backend services can be installed on several machines. But in this tutorial we will provide you with detailed instructions on how to install Lokit on a single one. Follow these guidelines if you want to perform a new, manual installation of the latest Lokit version.

#### Step1 : Requirements

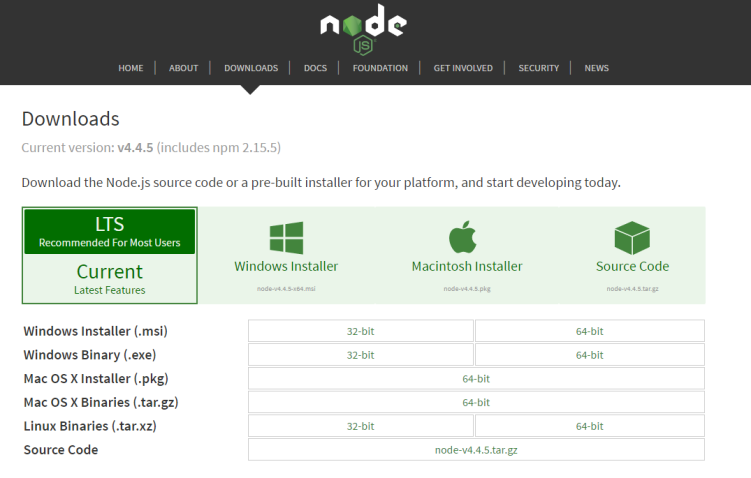
First-of-all you need to install GIT, .NET CORE and NodeJs on your machine. Follow the guides from the official websites to install them.



Install GIT ([link](https://git-scm.com/downloads))



Install .NET core ([link](https://www.microsoft.com/net/core#windows))



Install NodeJs ([link](https://nodejs.org/en/download/))

When the installation is finished, install the npm package “ember” on your machine and execute the command below in a command prompt :

npm install –g ember-cli

Now the required Softwares are installed on your machine, backend and frontend modules can be built with “dotnet or ember” command line instructions.

In the actual state the modules cannot be started because they required a database connection. An SQLSERVER **or** SQLITE database **and** Mongo database must be deployed and for all of them you need an account with “Create Table” and “Read/Insert/Update/Delete” permissions.

The accounts are used to migrate the tables and interact with them.

Three schemas need to be created on SqlServer and / or SqlLite : one for the authorization server, uma and configuration API. Reuse the schema name when you’re upading the connection string in the “appsettings.json” file for the steps 5 and X.

#### Step2 : Download sources

Fetch backend and frontend sources from the GIT repositories :

* Backend component : <https://github.com/thabart/SimpleIdentityServer.git>
* Frontend component : <https://github.com/thabart/UmaManagerWebSite.git>
* Frontend API component : <https://github.com/thabart/UmaManagerWebSiteApi.git>

When the sources have been retrieved, you should see two new repositories :

* SimpleIdentityServer : it contains all backend services
* UmaManagerWebSite : it contains the website
* UmaManagerWebSiteApi : contains API of the website

#### **Step3 : Build backend modules**

Open a command prompt and navigate to the directory where the GIT repositories have been downloaded. Execute the following commands to download all Nuget packages and build the modules :

cd SimpleIdentityServer\SimpleIdentityServer\src

dotnet restore

dotnet build SimpleIdentityServer.Startup

dotnet build SimpleIdentityServer.Manager.Host.Startup

dotnet build SimpleIdentityServer.Configuration.Startup

dotnet build SimpleIdentityServer.Manager.Host.Startup

#### Step 4 : Build frontend modules

Do the same than before and execute the following commands :

cd UmaManagerWebSite

ember build

If an error occurred, refer to the annex to try to fix it.

#### Step5 : Deploy databases

Now your solution is built and you can use an SqlServer or SqlLite account with “Create Table” permission, then the SQL tables can be migrated.

Do the same than before and navigate to the folder “SimpleIdentityServer\src\ SimpleIdentityServer.DataAccess.SqlServer” open the “appsettings.json” file and update its properties :

* Replace the ConnectionString with yours.
* If you’re using SqlServer then set the property values “isSqlServer” to true and “isSqlLite” to false, do the revert if you’re using SqlLite.

Launch the migration by executing the following command :

dotnet ef –f net46 database update

Repeat the previous step by replacing the folder “SimpleIdentityServer\src\SimpleIdentityServer.DataAccess.SqlServer” by “SimpleIdentityServer.Configuration.EF” and “SimpleIdentityServer.Uma.EF”

#### Step6 : Update backend modules configuration

Repeat the step 5 without executing the command line instruction for the projects :

* *SimpleIdentityServer\src\ SimpleIdentityServer.Startup*
* *SimpleIdentityServer\src\ SimpleIdentityServer.Manager.Host.Startup :* use the same configuration as “SimpleIdentityServer.Startup”
* *SimpleIdentityServer\src\SimpleIdentityServer.Configuration.Startup*
* *SimpleIdentityServer\src\ SimpleIdentityServer.Uma.Host*

#### Step7 : Run backend modules

Execute each instruction below in a new command prompt :

cd SimpleIdentityServer\src\SimpleIdentityServer.Startup

dotnet –f net46 –server.urls=http://\*:5000

cd SimpleIdentityServer\src\SimpleIdentityServer.Uma.Host

dotnet –f net46 –server.urls=http://\* :5001

cd SimpleIdentityServer\src\SimpleIdentityServer.Manager.Host.Startup

dotnet –f net46 –server.urls=http://\*:5002

cd SimpleIdentityServer\src\SimpleIdentityServer.Configuration.Startup

dotnet –f net46 –server.urls=http://\* :5004

#### Step 8 : Update frontend configuration modules

Open the folder “UmaManagerWebSiteApi” and with your preferred editor open the file : “server-api.js”. Update the default value of “db” variable with a connection string used to connect to your Mongo database. Save your changes and close the editor.

The authorization server, uma and Website API URLs are stored in the files “UmaManagerWebSiteApi\configuration.js” and “UmaManagerWebSite\config\environment.js”. If you’re using the default configuration, leave their values as default.

The table below list the URLs and their locations :

|  |  |  |
| --- | --- | --- |
| URL | Configuration files | Default value |
| Authorization server | UmaManagerWebSite \ config \ environment.js  UmaManagerWebSiteApi \ configuration.js | <http://localhost:5000> |
| Uma server | UmaManagerWebSiteApi \ configuration.js | <http://localhost:5001> |
| Manager API | UmaManagerWebSiteApi \ configuration.js | <http://localhost:5002> |
| Configuration API | UmaManagerWebSiteApi \ configuration.js | <http://localhost:5004> |
| WebSiteAPI | UmaManagerWebSite \ config \ environment.js | <http://localhost:8080> |

#### Step 9 : Run frontend modules

To run the API website execute the command :

cd UmaManagerWebSiteApi

node server-api.js

To run the website execute the following command :

cd UmaManagerWebSite

ember serve

### With docker

Deploy the solution with Docker is easier and faster.

1. Install Docker on your machine ([link](https://docs.docker.com/engine/installation/))
2. Open VirtualBox and add some redirection ports rules :
   1. Source : 8080, Destination : 8080
   2. Source : 4200, Destination: 4200
   3. Source : 5000, Destination: 5000
3. Fetch the sources from the GIT repository : <https://github.com/thabart/SimpleIdentityServerDocker.git>
4. Open a command prompt and navigate to the new folder
5. Execute the command : *docker-compose up* and wait-for the build !