## Configuring web farms on MVC applications

When using the MVC development model, your MVC application and Kentico application should always be configured to run as servers in a web farm (see Starting with MVC development). The web farm ensures that the MVC application invalidates cache according to content or setting changes made in the Kentico application and vice versa.

The MVC application works as a web farm server, because it uses the same database as the Kentico application and performs operations through the Kentico API.

Licensing of the web farm servers works automatically for basic scenarios – see Kentico licensing for MVC applications for details.

**Adding web farm servers to scale performance**

If you wish to use a web farm to scale the site's performance, you can add further instances of the MVC application. We recommend using the following process:

1. Develop and test the site in a web farm with two servers (one MVC application, one Kentico application).
2. Deploy any number of additional instances of the same MVC application. Each instance must connect to the same Kentico database.

The automatic web farm mode automatically registers the new instances as web farm servers and ensures correct synchronization (among all instances of the MVC application and the Kentico application).

If you need to scale the performance of the Kentico administration interface used to manage the site content and settings, you can also run multiple instances of the Kentico application in the web farm. In this scenario, use the standard approach for configuring web farm servers. You need to use one of the servers as the "primary" Kentico instance (for example for holding files shared by the entire web farm, such as locally stored search indexes).

**Note**: The above scenarios require a license that supports the additional number of web farm servers.

## Caching in MVC applications

You can cache the data and output of controller actions in your MVC application.

**Prerequisite**: For caching in your MVC application to work correctly, your Kentico and MVC applications need to be configured as a web farm.

Specifically, you can cache the following:

- Data you retrieve to display on individual web pages.
- Output of controller actions.

### Caching data in MVC applications

We recommend using caching of data, especially for data that is frequently accessed (queried from the database). For example, you may want to cache individual pages, forms and any other data that you retrieve from the Kentico database and display on the site.

For pages, custom table data or forms, you can handle the caching in individual repositories that make use of the generated providers. Another approach would be using AOP and decorating the individual repositories – see our MVC demo site for a reference on how to use this approach.

Use the **CacheHelper.Cache** method to cache data in your code.

### Examples

**Notes**

- To ensure consistency of the viewed data, always generate cache keys with names that include all the parameters that you use to retrieve the data. For example, a culture code variable in the cache key for page data ensures that visitors do not see cached articles displayed in an old culture after switching to a new culture.
- The examples use minimal cache dependencies. See Setting cache dependencies for more information on how to configure cache dependencies for your scenarios.

**Caching the data of multiple articles**

```
using System;
using System.Collections.Generic;
using System.Linq;

using CMS.Helpers;
using CMS.SiteProvider;

...

public IEnumerable<Article> GetArticles(int count = 0)
{
    string culture = "en-us";
    string siteName = SiteContext.CurrentSiteName;

    Func<IEnumerable<Article>> dataLoadMethod = () => ArticleProvider.GetArticles()
            .OnSite(siteName)
            .Culture(culture)
            .TopN(count)
            .OrderByDescending("DocumentPublishFrom")
            .TypedResult; // Ensures that the result of the query is saved, not the
query itself

    var cacheSettings = new CacheSettings(10, "myapp|data|articles", siteName,
culture, count)
    {
        GetCacheDependency = () =>
        {
            // Create caches dependencies. This example makes the cache clear data
when any article is modified, deleted, or created in Kentico.
            string dependencyCacheKey = String.Format("nodes|{0}|{1}|all", siteName,
Article.CLASS_NAME.ToLowerInvariant());
            return CacheHelper.GetCacheDependency(dependencyCacheKey);
        }
    };

    return CacheHelper.Cache(dataLoadMethod, cacheSettings);
}
```

**Caching the data of a single article**

```
using System;
using System.Linq;

using CMS.Helpers;
using CMS.SiteProvider;

...

public Article GetArticle(Guid nodeGuid)
{
    string culture = "en-us";
    string siteName = SiteContext.CurrentSiteName;

    Func<Article> dataLoadMethod = () => ArticleProvider.GetArticle(nodeGuid, culture,
siteName)
                                                        .TopN(1)
                                                        .FirstOrDefault();

    var cacheSettings = new CacheSettings(10, "myapp|data|article", nodeGuid, culture,
siteName)
    {
        GetCacheDependency = () =>
        {
            // Creates cache dependencies. This example makes the cache clear the data
when the specified article is modified in Kentico (any culture version).
            string dependencyCacheKey = String.Format("nodeguid|{0}|{1}", siteName,
nodeGuid);
            return CacheHelper.GetCacheDependency(dependencyCacheKey);
        }
    };

    return CacheHelper.Cache(dataLoadMethod, cacheSettings);
}
```

## Caching controller action output in MVC applications

You can cache the output of individual controller actions using ASP.NET output caching. This can significantly increase your website performance. We recommend caching the output of all possible controller actions, especially on controllers that return often requested views.

Use the OutputCache attribute to mark the action methods you want to cache. Set the cache duration (in seconds) and other properties of the attribute.

When storing the output cache on the server (when the *OutputCache* attribute's *Location* property is set to *Server*), you can define cache dependencies to accommodate various scenarios with both static and dynamic content. Use the **HttpContext.Response.AddCacheItemDependency()** method to add dependencies in individual controller action methods.

Controller actions can be called in situations in which the dummy cache items have not yet been created. To account for that, ensure the existence of the dummy cache item explicitly by calling *CacheHelper.EnsureDummyKey()*.

```
using System;
using System.Web.UI;
using System.Web.Mvc;

using CMS.Helpers;

...

[OutputCache(Duration=600, VaryByParam="none", Location = OutputCacheLocation.Server)]
public ActionResult Index()
{
        var articles = mArticleRepository.GetArticles();

        // Sets cache dependencies. This example makes the system clear the cache when
any article is modified in Kentico.
        string dependencyCacheKey = String.Format("nodes|mvcsite|{0}|all", Article.
ClassName.ToLowerInvariant());
        CacheHelper.EnsureDummyKey(dependencyCacheKey);
        HttpContext.Response.AddCacheItemDependency(dependencyCacheKey);

        return View(articles);
}
```

**Note**: The example uses minimal cache dependencies – the cache is cleared when any of the articles the method works with is modified, deleted, or when a new article is created. See Setting cache dependencies for more information on how to configure cache dependencies for your scenarios.