Cloud design patterns

Challenges in cloud development (1)

- Availability

Availability is the proportion of time that the system is functional and working, usually measured as a percentage of uptime. It can be affected by system errors, infrastructure problems, malicious attacks, and system load. Cloud applications typically provide users with a service level agreement (SLA), so applications must be designed to maximize availability.

- Data Management

Data management is the key element of cloud applications, and influences most of the quality attributes. Data is typically hosted in different locations and across multiple servers for reasons such as performance, scalability or availability, and this can present a range of challenges. For example, data consistency must be maintained, and data will typically need to be synchronized across different locations.

- Design and Implementation

Good design encompasses factors such as consistency and coherence in component design and deployment, maintainability to simplify administration and development, and reusability to allow components and subsystems to be used in other applications and in other scenarios. Decisions made during the design and implementation phase have a huge impact on the quality and the total cost of ownership of cloud hosted applications and services.

Challenges in cloud development (2)

- Messaging

The distributed nature of cloud applications requires a messaging infrastructure that connects the components and services, ideally in a loosely coupled manner in order to maximize scalability. Asynchronous messaging is widely used, and provides many benefits, but also brings challenges such as the ordering of messages, poison message management, idempotency, and more.

- Management and Monitoring

Cloud applications run in a remote datacenter where you do not have full control of the infrastructure or, in some cases, the operating system. This can make management and monitoring more difficult than an on-premises deployment. Applications must expose runtime information that administrators and operators can use to manage and monitor the system, as well as supporting changing business requirements and customization without requiring the application to be stopped or redeployed.

Challenges in cloud development (3)

- Performance and Scalability

Performance is an indication of the responsiveness of a system to execute any action within a given time interval, while scalability is ability of a system either to handle increases in load without impact on performance or for the available resources to be readily increased. Cloud applications typically encounter variable workloads and peaks in activity. Predicting these, especially in a multitenant scenario, is almost impossible. Instead, applications should be able to scale out within limits to meet peaks in demand, and scale in when demand decreases. Scalability concerns not just compute instances, but other elements such as data storage, messaging infrastructure, and more.

Challenges in cloud development (4)

- Resiliency

Resiliency is the ability of a system to gracefully handle and recover from failures. The nature of cloud hosting, where applications are often multitenant, use shared platform services, compete for resources and bandwidth, communicate over the Internet, and run on commodity hardware means there is an increased likelihood that both transient and more permanent faults will arise. Detecting failures, and recovering quickly and efficiently, is necessary to maintain resiliency.

- Security

Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. Cloud applications are exposed on the Internet outside trusted on-premises boundaries, are often open to the public, and may serve untrusted users. Applications must be designed and deployed in a way that protects them from malicious attacks, restricts access to only approved users, and protects sensitive data.

Health Endpoint Monitoring.

- **Health Endpoint Monitoring**. This example shows how you can set up a web endpoint that checks the health of dependent services by returning appropriate status codes.
- The endpoints are designed to be consumed by a watchdog monitoring service such as Windows Azure Endpoint Monitoring, but you can open and invoke the endpoint operations from a browser to see the results.
- You can also deploy and configure your own endpoint monitoring tool of choice to send requests to the service operations and analyze the responses received.

Health Endpoint Monitoring.

- Start the example running and navigate to one of the monitoring endpoints. Use the F12 Developer Tools in Internet Explorer or the equivalent in your browser to watch the network traffic response pattern.
- The response will be 200 (OK), or a 500 error code that indicates a problem with a service.

Health Endpoint Monitoring.

- The sample contains four endpoints that you can test:
  - **/HealthCheck/CoreServices** This is a simple check on dependent services to determine if they are responding.
  - **/HealthCheck/ObscurePath/{*path*}** This demonstrates a check operation that uses a configurable obscure path defined in the service configuration file. Replace *path* with the value of the setting named Test.ObscurePathfrom in the file ServiceConfiguration.*.csfg. Note that this technique is not to be used as an alternative to properly securing an application and implementing authentication.

Health Endpoint Monitoring.

- **/HealthCheck/CheckUnstableServiceHealth** This will randomly return a 500 exception for approximately 20% of the requests.
- **/HealthCheck/TestResponseFromConfig** This returns a response code set in configuration (.cscfg) for testing. Deploy the solution with endpoint monitoring configured to use this operation, and then explicitly set the "Test.ReturnStatusCode" setting to return the required status code to test and demonstrate the effects of these codes on the monitoring service.

Health Endpoint Monitoring.

- How to access an endpoint that's secured using authentication. Not all tools and frameworks can be configured to include credentials with the health verification request. For example, Microsoft Azure built-in health verification features can't provide authentication credentials. Some third-party alternatives are Pingdom, Panopta, NewRelic, and Statuscake.

Health Endpoint Monitoring.

When to use this pattern

- Monitoring websites and web applications to verify availability.
- Monitoring websites and web applications to check for correct operation.
- Monitoring middle-tier or shared services to detect and isolate a failure that could disrupt other applications.
- Complementing existing instrumentation in the application, such as performance counters and error handlers. Health verification checking doesn't replace the requirement for logging and auditing in the application. Instrumentation can provide valuable information for an existing framework that monitors counters and error logs to detect failures or other issues. However, it can't provide information if the application is unavailable.

Static Content Hosting Pattern

- Web applications typically include some elements of static content. This static content may include HTML pages and other resources such as images and documents that are available to the client, either as part of an HTML page (such as inline images, style sheets, and client-side JavaScript files) or as separate downloads (such as PDF documents).
- Although web servers are well tuned to optimize requests through efficient dynamic page code execution and output caching, they must still handle requests to download static content. This absorbs processing cycles that could often be put to better use.

Static Content Hosting Pattern

- In most cloud hosting environments it is possible to minimize the requirement for compute instances (for example, to use a smaller instance or fewer instances), by locating some of an application's resources and static pages in a storage service. The cost for cloud-hosted storage is typically much less than for compute instances.
- When hosting some parts of an application in a storage service, the main considerations are related to deployment of the application and to securing resources that are not intended to be available to anonymous users.

Static Content Hosting Pattern

- **Issues and Considerations**
- Consider the following points when deciding how to implement this pattern:
- • The hosted storage service must expose an HTTP endpoint that users can access to download the static resources. Some storage services also support HTTPS, which means that it is possible to host resources in storage service that require the use of SSL.
- • For maximum performance and availability, consider using a content delivery network (where available) to cache the contents of the storage container in multiple datacenters around the world. However, this will incur additional cost for the use of the content delivery network.
- • Storage accounts are often geo-replicated by default to provide resiliency against events that might impact a datacenter. This means that the IP address may change, but the URL will remain the same.

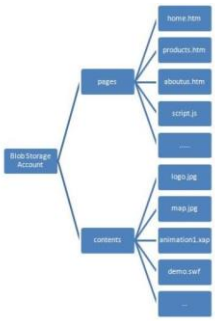
Static Content Hosting Pattern

- **When to Use this Pattern**
- This pattern is ideally suited for:
- • Minimizing the hosting cost for websites and applications that contain some static resources.
- • Minimizing the hosting cost for websites that consist of only static content and resources. Depending on the capabilities of the hosting provider's storage system, it might be possible to host a fully static website in its entirety within a storage account.
- • Exposing static resources and content for applications running in other hosting environments or on-premises servers.
- • Locating content in more than one geographical area by using a content delivery network that caches the contents of the storage account in multiple datacenters around the world.
- • Monitoring costs and bandwidth usage. Using a separate storage account for some or all of the static content allows the costs to be more easily distinguished from hosting and runtime costs.

Static Content Hosting Pattern

- This pattern might not be suitable in the following situations:
- • The application needs to perform some processing on the static content before delivering it to the client. For example, it may be necessary to add a timestamp to a document.
- • The volume of static content is very small. The overhead of retrieving this content from separate storage may outweigh the cost benefit of separating it out from the compute resources.

Static Content Hosting



Static Content Hosting Pattern

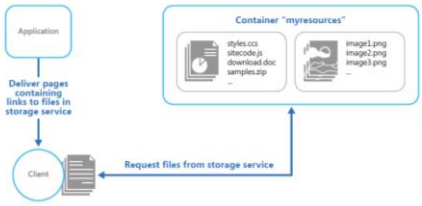


FIGURE 1
Delivering static parts of an application directly from a storage service

Static Content Hosting Pattern

- The links in the pages delivered to the client must specify the full URL of the blob container and resource. For example, a page that contains a link to an image in a public container might contain the following.
- **HTML**
- <img src="http://mystorageaccount.blob.core.windows.net/myresources/image1.png"
- alt="My image" />

Static Content Hosting Pattern

- The **StaticContentHosting.Cloud** project contains configuration files that specify the storage account and container that holds the static content.
- **XML**
- <Setting name="StaticContent.StorageConnectionString"
- value="UseDevelopmentStorage=true" />
- <Setting name="StaticContent.Container" value="static-content" />

Static Content Hosting Pattern

Settings.cs of the StaticContentHosting.Web project

- methods to extract these values and build a string value containing the cloud storage account container URL.

```
public class Settings {
public static string StaticContentStorageConnectionString {
get { return RoleEnvironment.GetConfigurationSettingValue(
"StaticContent.StorageConnectionString"); } }
public static string StaticContentContainer {
get { return RoleEnvironment.GetConfigurationSettingValue("StaticContent.Container"); } }
public static string StaticContentBaseUrl {
get { var account = CloudStorageAccount.Parse(StaticContentStorageConnectionString);
return string.Format("{0}/{1}", account.BlobEndpoint.ToString().TrimEnd('/'),
StaticContentContainer.TrimStart('/'));
} }}
```

Static Content Hosting Pattern

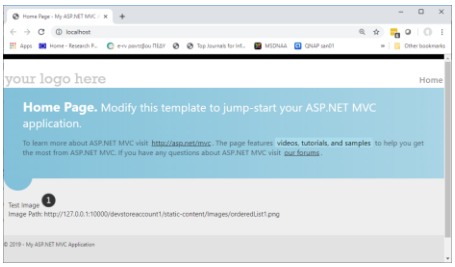
- The **StaticContentUrlHtmlHelper** class in the file StaticContentUrlHtmlHelper.cs exposes a method named **StaticContentUrl** that generates a URL containing the path to the cloud storage account if the URL passed to it starts with the ASP.NET root path character (**~**).

```
public static class StaticContentUrlHtmlHelper{
public static string StaticContentUrl(this HtmlHelper helper, string contentPath) {
if (contentPath.StartsWith("~")) {
contentPath = contentPath.Substring(1); }
contentPath = string.Format("{0}/{1}", Settings.StaticContentBaseUrl.TrimEnd('/'),
contentPath.TrimStart('/'));
var url = new UrlHelper(helper.ViewContext.RequestContext);
return url.Content(contentPath); }}
```

Static Content Hosting Pattern

- The file Index.cshtml in the Views\Home folder contains an image element that uses the **StaticContentUrl** method to create the URL for its **src** attribute.
- **HTML**
-

Static Content Hosting Pattern



Static Content Hosting Pattern

- **Related Patterns and Guidance**
- The following pattern may also be relevant when implementing this pattern:
- • **Valet Key Pattern**. If the target resources are not supposed to be available to anonymous users it is necessary to implement security over the store that holds the static content. The Valet Key pattern describes how to use a token or key that provides clients with restricted direct access to a specific resource or service such as a cloud-hosted storage service.