Windows Azure ile Cloud Computing Uygulamaları – 8

Zaman sürekli ilerliyor. Geçmişte büyük prodüksiyonlar ile yapılacağı inanılan çalışmalar, kişisel cihazlar ile gerçekleştirilebilir hale geldi. Yaşanılan anları kayıt etme isteği, herkesi fotoğrafçısı ve rejisörü olabilme yollu açmıştır.

Yaşam hızlı değişiyor. Yaşanan değişimin en büyük mimarı ise teknoloji olarak görünmektedir. Teknoloji kullanıcılarına birçok olanak sağladığı gibi kendisi içerisinde farklı olanaksızlar da bulunmaktadır.

Şirketler, iş süreçlerini standartlar ile devamlılığını sağlayabilmek amacı ile çeşitli teknolojiler kullanılmaktadır. Kullanılan teknolojiler, gerçekleştirilmesi istenen sürecin başarılı ve en az problem ile tamamlaması beklenmektedir. Beklenen sonuçların alınması, müşteri memnuniyetinin sağlanması şirketler için önemli olmaktadır.

İş süreçleri devamlı ve istenilen kalite de yürütülmesi, müşteri ve sağlayıcı(üretici) arasında olumlu ilişkilerin kurulabilmesi için önemli bir bağ olarak görülmektedir. Şirketler sundukları hizmet ya da ürünlerin, müşterilerine tanıtabilmek amacı ile reklam filmleri ve kataloglar kullanmaktadır. Kullanılan metreyeler, teknoloji sistemlerin kullanılması ile oluşturularak müşterilere ulaştırılmaktadır.

Günümüzde şirketler tanıtımlarını sağlayabilmek amacı ile birçok yöntem kullanmaktadır. Kullanılan yöntemlerin büyük kısmı, teknoloji sistemlerin kullanıldığı ve müşteri memnuniyetin ön planda olduğu sistemlerdir. Örneğin; Atın ve mücevher satışı yapan şirketler, hazırladığı ürünleri Web sitesi üzerinden çeşitli fotoğraflar kullanarak tanıtımını yapmaktadır. Ayrıca her zaman müşterilerinin yanında olabilme olanağı, ürün destek ve satış sürecini için aktif süreçler gerçekleştirebilmektedir. Gerçekleşen süreçler ile şirket, genişleyen müşteri portföyüne sahip olmaktadır.

İş dünyasının hızlı manevrana cevap verebilmek, şirketlerin yaşamının devam ettirebilmesi için çok önemlidir. Genişleyen iş süreçleri ile yeni teknoloji **inovasyon**ların önü açılmaktadır. **İnovasyon** hareketleri, geçmişte yapılan hatalarda ders alınarak oluşturulmaktadır. Günümüzün **inovasyon**

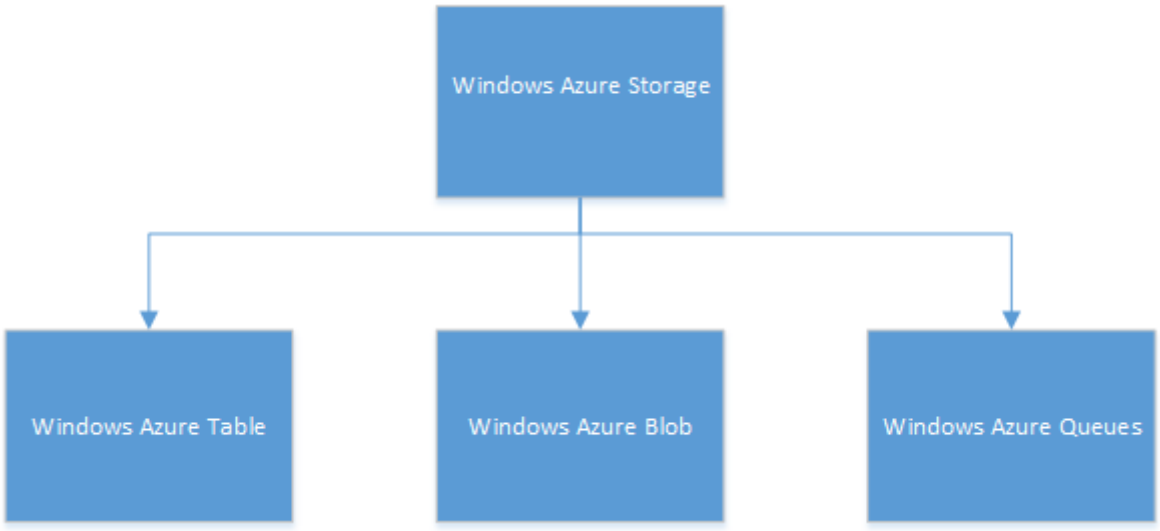
çözümleri incelendiğinde en belirgin özellikleri sürdürülebilir, ekonomik, esnek ve güvenilir olarak sıralanmaktadır. Belirtilen özellikler **Cloud Computing** mimarisinin en temel özellikleridir.

Cloud Computing mimarisi çeşitli şirketler tarafından yorumlanmaktadır. Özellikle Microsoft, **Windows Azure Platform** ürünü ile kullanıcılarına ekonomik ve kolay yönetilebilir hizmetler sağlamaktadır.

Windows Azure Platform, esnek ve devamlılığı sağlanması istenen birçok iş çözümün yaşayabilmesi için çeşitli çözümleri sunmaktadır. Sunulan iş çözümleri, yeni birçok konsept kullanarak, iş süreçleri esnek olabilmesine olanak sağlamaktadır.

Şirketler gerçekleştirdiği yoğun iş süreçleri, teknoloji olarak incelendiğin de problemlerin temelinde depolama sorunları ile karşılaşılmaktadır. Sürecin güvenli ve en az problem ile sağlanabilmesi amacı ile Microsoft, **Windows Azure Platform** ile yeni depolama konseptti, **Windows Azure Storage** altyapısını sunmuştur.

Windows Azure Storage, yüksek optimizasyonu değeri olan ve yüksek güvenliği seviyeli depolama altyapısıdır. Söz konusu altyapı, klasik uygulama geliştirme yaklaşımın ötesin de nesnel ve veri kayıp problemlerin en alt düzey de olan bir sistemdir. **Windows Azure Storage** altyapısı, iş gereksinimlerine cevap üretecek üç parça dan oluşmaktadır. Aşağıda **Windows Azure Storage** altyapısı parçaları şema ile gösterilmektedir.

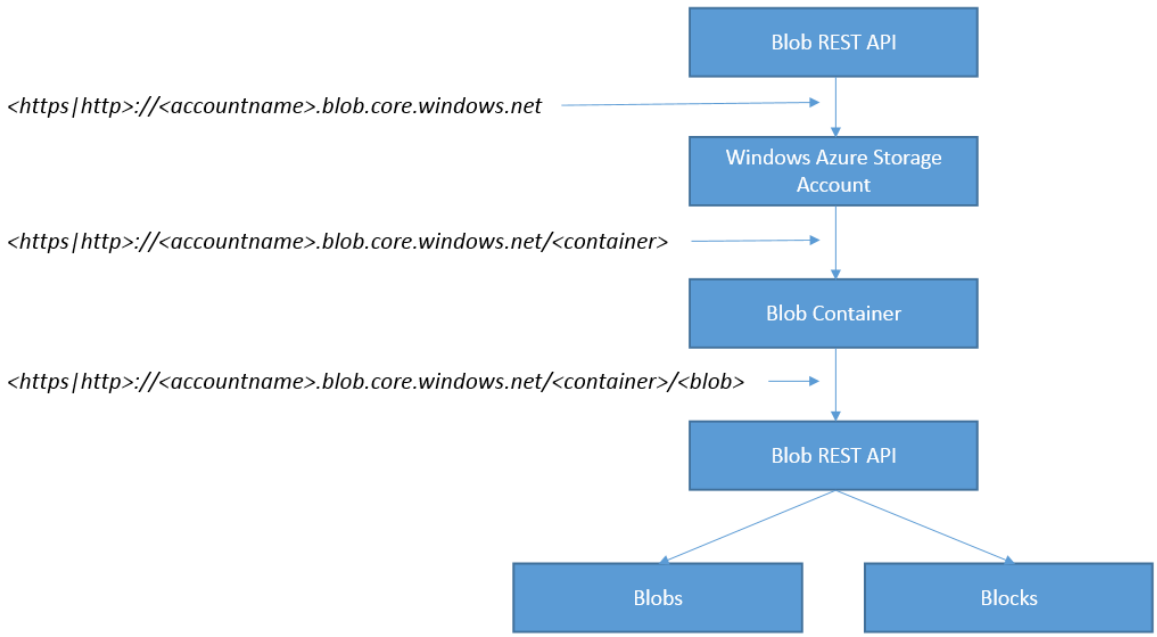


Windows Azure Storage, kullanıcı iş gereksinimlerine göre **Table**, **Queue** ve **Blob** isimli üç parçadan oluşmaktadır. **Windows Azure Storage**, **Table** ve **Queue** parçaları ile ilgili "***Windows Azure ile Cloud Computing Uygulamaları***" makale serisinin diğer bölümlerin de bilgiler verilmiştir. Bu çalışma ile Windows Azure Blob Storage ile ilgili bilgiler verilmesi amaçlanmaktadır.

İş uygulamaları çoğunlukla kullanıcıların dan çeşitli Binary(fotoğraf ya da Word dosyası) içerikler alınmaktadır. Klasik uygulama geliştirme yaklaşımı incelendiğin de kullanıcıların dan alınan dosya içeriklerinin uygulama makinesi üzerine konumlandırılmaktadır. Kullanıcı dosyaların ve iş uygulaması ile aynı makine üzerine konumlandırılması, zaman içerisinde depolama çeşitli problemlerin ortaya çıkmasına neden olmaktadır.

İş uygulamalarının, **Windows Azure Blob Storage** altyapısı kullanarak geliştirilmesi ile uygulama verilerin depolama yaklaşımı farklı bir konseptte taşınmaktadır. Uygulama verileri, **Windows Azure**

Storage ile uygulama alanından bağımsız, esnek ve yüksek güvenlikli altyapıya taşımaktadır. Aşağıda **Windows Azure Blob Storage** ile ilgili iletişim şeması bulunmaktadır.



Windows Azure Blob Storage, iş gereksinimlerine göre çeşitli olanaklar sunmaktadır. Yukarıda bulunan şema da görüldüğü gibi **Windows Azure Storage**, **REST** mimari ile iletişimini sağlayan, iki ayrı parçadan oluşmaktadır. Aşağıda **Windows Azure Blob Storage** altyapısı ile ilgili bazı özellikler ve açıklamaları bulunmaktadır.

Özellik	Açıklama
Snapshot	Depolanan verinin belirlenen tarihe kadar sadece okunabilir halde dondurulması şeklinde hizmet verme durumudur.
Erişim Seviyesi	Depolanması istenen verinin özel(private), sadece genel(public) okuma ya da genel(public) erişim ile tüm yetkiler sağlanabilmesi şeklinde erişim yetkileri sınırlandırılabilmektedir.
Disaster Recovery	Geo-repolaction özelliği sayesinde depolanan veri bir den fazla lokasyon üzerine kopyalanmaktadır. Gerçekleşen kopyalama ile herhangi noktada yaşanan sistem problemlerin de farklı bir nokta üzerinde veri erişim süreçleri devam ettirilebilmektedir. Sağlanan olanaklar ile Disaster Recovery sürecine destek verilebilmektedir.
Backup	**Windows Azure Blob Storage** içerisinde depolanan veriler, çeşitli **Windows Azure** hizmetleri ile yedekle olanakları sağlanabilmektedir.
Geo-Replication	Depolanan verinin farklı coğrafik nokta da kopyaların oluşturulabilmektedir.

Erişim Protokolleri	**Windows Azure Blob Storage**, üzerinde taşıdığı verileri Http ya da Https olarak erişilebilmesinin olanak sağlayabilmektedir. Depolama sözümü 2616 RPC standart da bağlı olarak Http ya da Https üzerinde REST iletişimine de olanak sağlamaktadır.
Logging	**Windows Azure Blob Storage** üzerinde yapılan işlemlerin geçmiş dönük incelebilmesi amacı ile yapılan veri hareketleri depolanabilmektedir.
En Fazla Depolama Miktarı	**Windows Azure Blob Storage**, **Block Blob** olarak en fazla 200GB ve **Page Blob** olarak ise, 1TB veri depolayabilmektedir.

Geliştirilen iş uygulamaları, gerçekleştireceği iş sürecine bağlı olarak, depolama planlamaları yapılmaktadır. Yapılan depolama planları, iş verisinin okuma / yazma yoğunlukları ya da veri boyutları ile ilgili şekillenmektedir. **Windows Azure Blob Storage** konsepti ile iş gereksinimlerine göre **Page Blob** ve **Block Blob** isimli iki farklı nesne ile hizmet vermektedir. Aşağıda söz konusu nesneler ile ilgili temel bilgi karşılaştırmaları bulunmaktadır.

Özellik	**Block Blob**	**Page Blob**
En fazla Veri Depolama	4MB	512Byte
En Az Veri Depolama	200GB	1TB
Snapshot	Destekliyor	Destekliyor
Erişim Seviyesi	Private,Public blob, Public container	Private,Public blob, Public container
Erişim Protokolleri	Http/Https/Rest	Http/Https/Rest
Logging	Destekliyor	Destekliyor

Yapılan anlatımın anlaşılması amacı ile "***WindowsAzure.FunnyApp***" isimli uygulama örneği hazırlanmıştır. Hazırlanan uygulama örneği ile iş senaryoları incelenerek, **Windows Azure Platform** hakkında farklı deneyimler paylaşılmıştır.

Uygulama örneğin de **Windows Azure Storage** ile ilişkili nesnelerin kullanımı kolaylaştırıcı anahtar değişkenler tanımlanmıştır. Tanımlanan değişkenler **Windows Azure Storage** içerisinde kullanılan nesnelerin yönetimini kolaylaştırması amaçlanmıştır.

```
public class Utils
{
    public const string ConfigurationString = "DataConnectionString";

    public const string CloudQueueKey = "imagequeue";
    public const string CloudBlobKey = "imageblob";
}
```

Uygulama Senaryosu - |

Kullanıcı tarafından yayınlanması istenen fotoğraf ile ilgili bilgileri girilerek, içeriğin uygulamaya yükleme süreci gerçekleşmektedir. Yükleme süreci kullanıcı tarafından fotoğrafın yüklenmesi ile **Windows Azure Table Storage** üzerine ilgili bilgilere yazılarak ve fotoğraf içeriğinin **Windows Azure Blob** Storage üzerine yüklenmesi ile tamamlanmaktadır. Gerçekleşen süreçler ile ilgili kaynak kodlar ve yorumları aşağıda bulunmaktadır.

```
public partial class ImageUploadPage : Page
{
    private static readonly object _look = new object();
    private static bool _storageInitialized = false;

    // işlem gerçekleştirlecek olan nesnelerin tanımlanması
    private static CloudBlobClient _blobClient;
    private static CloudQueueClient _queueClient;

    protected void Page_Load(object sender, EventArgs e)
    {
        this.Page.Title = "Image Uploads";
        if (IsPostBack) return;
        InitializeStorage();
    }

    protected void ButtonSave_Click(object sender, EventArgs e)
    {
        if (FileUploadImage.HasFiles & Page.IsValid)
        {
            // Blob nesnesine yüklenecek dosya için benzersiz bir isim oluşturul
ması işlmei
            string uniqueBobName = string.Format("{0}/funnyimage_{1}{2}", Utils.
CloudBlobKey,
                                                 Guid.NewGuid().ToString(),
                                                 Path.GetExtension(FileUploadIma
ge.FileName));

            // Blob nesnesi ile ilgili anahtar ve nesne örneğinin alınması
            CloudBlockBlob blob = _blobClient.GetBlockBlobReference(uniqueBobNam
e);

            // Blob nesnesi üzerine yüklenecek, Blob nesnesine dosyasının türünü
n atanması
            blob.Properties.ContentType = FileUploadImage.PostedFile.ContentType
;

            // Blob nesnesine dosyasının yüklenmesi
            blob.UploadFromStream(FileUploadImage.FileContent);

            FunnyAppRepository<Post> postRepository = new FunnyAppRepository<Pos
t>();
            FunnyAppRepository<Tag> tagRepository = new FunnyAppRepository<Tag>(
);

            MembershipUser user = Membership.GetUser(Page.User.Identity.Name);
            if (user != null)
            {
                Post post = new Post
                    {
                        PostContent = TextBoxDescription.Text,
                        PostTitle = TextBoxTitle.Text,
                        State = false,
                        UserId = user.ProviderUserKey.ToString()
                    };

                string[] tags = TextBoxTag.Text.Split(';');
                foreach (string tag in tags)
                {
```

```
                    if (!string.IsNullOrEmpty(tag))
                    {
                        tagRepository.Create(new Tag()
                            {
                                PostRowKey = post.RowKey,
                                PostPartitionKey = post.PartitionKey,
                                TagName = tag,
                            });
                        tagRepository.SubmitChange();
                    }
                }

                postRepository.Create(post);
                postRepository.SubmitChange();

                // Kuyruk nesneleri
                CloudQueue queue = _queueClient.GetQueueReference(Utils.CloudQue
ueKey);
                // mesaj içeriğinin oluşturulması
                // mesaj içerisinden birden fazla bilgi olması sebebi ile "," ka
rakteri
                // ile bilgiler birbirinden ayrılmıştır.
                CloudQueueMessage message =
                    new CloudQueueMessage(string.Format("{0},{1},{2}", blob.Uri,

                        post.PartitionKey, post.RowKey));
                // Mesaj kuyruğa eklenmiştir.
                queue.AddMessage(message);

                LabelResult.Text = "Uploaded";
            }
            else
            {
                LabelResult.Text = "Failed";
            }
        }
    }

    private void InitializeStorage()
    {
        if (_storageInitialized)
        {
            return;
        }

        lock (_look)
        {
            if (_storageInitialized)
            {
                return;
            }

            try
            {
                // hesap bilgilerinin alınması
                CloudStorageAccount storageAccount =
                    CloudStorageAccount.FromConfigurationSetting(Utils.Configura
tionString);
```

```
                // image blob taşıyıcısının oluşturulması
                _blobClient = storageAccount.CreateCloudBlobClient();
                CloudBlobContainer container =
                    _blobClient.GetContainerReference(Utils.CloudBlobKey);
                container.CreateIfNotExist();

                // Blob taşıyıcısına ile ilgili erişim ayarlarının tanımlanması
                var permissions = container.GetPermissions();
                permissions.PublicAccess = BlobContainerPublicAccessType.Contain
er;
                container.SetPermissions(permissions);

                // create queue to communicate with worker role
                _queueClient = storageAccount.CreateCloudQueueClient();
                CloudQueue queue = _queueClient.GetQueueReference(Utils.CloudQue
ueKey);
                queue.CreateIfNotExist();
            }
            catch (WebException exception)
            {
                Trace.Write(exception.Message);
            }

            _storageInitialized = true;
        }
    }
}
```

Uygulama Senaryosu - ||

Uygulama örneği olan "***WindowsAzure.FunnyApp***" çalışması, kullanıcısından aldığı fotoğraf içerikleri yeninde boyutlandırarak, yayınlamaktadır. Yapılan boyutlandırma işlem yoğunluğun dengelemek ve kullanıcı işlem sürecinin gerçekleştirebilmesi amacı ile **Worker Role** tasarlanmıştır. Tasarlanan **Worker Role**, **Windows Azure Blob Storage** üzerinde boyutlandırılması amaçlanan dosya okunarak, boyutlandırma işlemlerine tabi tutulmaktadır. Yapılan işlemler sonucunda içerik URL adresi oluşmaktadır. Gerçekleşen süreçler ile ilgili kaynak kodlar ve yorumları aşağıda bulunmaktadır.

```
public class WorkerRole : RoleEntryPoint
{
    // işlem gerçekleştirlecek olan nesnelerin tanımlanması
    private CloudQueue _queue;
    private CloudBlobContainer _container;

    public override void Run()
    {
        // Worker nesnesi içerisinde sürecin sürekli olarak gerçekleştirilmesi a
macı ile
        // sonsuz döngüye alınması gerekmektedir.
        while (true)
        {
            try
            {
                // İşlem sürecinde kuyruk içerisinde bulunan mesajın dinleme işl
emi
                CloudQueueMessage message = _queue.GetMessage();
                if (message != null)
                {
```

```
                    string[] messageArray = message.AsString.Split(new char[] {
','  });
                    string outputBlobUri = messageArray[0];
                    string partitionKey = messageArray[1];
                    string rowkey = messageArray[2];

                    // Kuyruk içerisinde okunan mesajı, döngü içerisinde tekrar
okuması önemek amacı ile
                    // mesaj Peek edilmektedir.
                    _queue.PeekMessage();

                    // kuyruk nesnesi üzerinde gelen Blob nesne adresinin
                    // istenilen formatlarda filitrelenmesi
                    string inputBlobUri =
                        Regex.Replace(outputBlobUri, "([^\\.]+)(\\.[^\\.]+)?$",
"$1-myimage$2");

                    // İşlem yapılması istenen Blob nesnenin kontrol eldilmesi
                    _container.CreateIfNotExist();

                    // okuma ve yazma süreçlerin gerçekleşen olan Blob adresleri
ne bağlantıların oluşturulması
                    CloudBlob inputBlob = _container.GetBlobReference(outputBlob
Uri);
                    CloudBlob outputBlob = _container.GetBlobReference(inputBlob
Uri);

                    // Blob nesnesi içerisinde bulunan resim içeriği okunması
                    using (BlobStream input = inputBlob.OpenRead())
                    using (BlobStream output = outputBlob.OpenWrite())
                    {
                        ProcessImage(input, output);

                        // Blob nesnesi üzerinde çalışılan içerik ile ilgili Blo
b üzerine etki etmesi işlemi
                        output.Commit();
                        // İşlem sonucu olarak üretilen içerisinde output Blob
                        // içerik türünün atanması
                        outputBlob.Properties.ContentType = "image/jpeg";
                        // İşlem sonucu olan içeriğin attanan özellikleri Blob n
esnesi üzerine etki etmesi
                        outputBlob.SetProperties();

                        FunnyAppRepository<Post> postRepository = new FunnyAppRe
pository<Post>();
                        Post post = postRepository.Find(partitionKey, rowkey);

                        // Boyutlandırma işlemine uğrayan fotoğraf içeriğin
                        // ilgili entity nesnesine yansıtılması
                        post.PostImage = outputBlobUri;
                        post.State = true;
                        postRepository.Update(post);
                        postRepository.SubmitChange();

                        _queue.DeleteMessage(message);
                    }
                }
            }
            catch (StorageClientException e)
```

```
            {
                Trace.Write(e);
            }
        }
    }

    public override bool OnStart()
    {
        // Worker üzerinde başlatılması amaçlanan varsayılan bağlantı sayısı
        ServicePointManager.DefaultConnectionLimit = 12;
        // Uygulama içerisinde bağlantı bilgilerinin alınma işlemi
        CloudStorageAccount.SetConfigurationSettingPublisher((configName, config
Setter) =>
            configSetter(RoleEnvironment.GetConfigurationSettingValue(configName
)));

        // Uygulama içerisinde bağlantı bilgilerinin alınma işlemi
        var storageAccount =
            CloudStorageAccount.FromConfigurationSetting(Utils.ConfigurationStri
ng);

        // kuyruk nesnesi ile ilgili refeasnların oluşturulması
        CloudQueueClient queueStorage = storageAccount.CreateCloudQueueClient();
        _queue = queueStorage.GetQueueReference(Utils.CloudQueueKey);

        // kuyruk nesnesinin bulunup-bulunmadığı kontrollünün yapılması
        _queue.CreateIfNotExist();

        // Blob nesnesi ile ilgili referasların oluşturulması
        CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();
        _container = blobStorage.GetContainerReference(Utils.CloudBlobKey);

        // Blob nesnesinin blunup-bulunmadığının kontrollünün yapılması
        _container.CreateIfNotExist();

        return base.OnStart();
    }

    // Resim boyutlandırma işlemleri ile fonksiyon
    public void ProcessImage(Stream input, Stream output) {. . .}
}
```

Windows Azure Blob Storage altyapısı, kullanımı ile ilgili iş süreçleri incelenmiştir. **Windows Azure Blob Storage** altyapısı **Page Blob** ve **Block Blob** depolama konsepti ile iş süreçlerine cevap üretmektedir. Depolama konseptleri, iş uygulamalarının, iş verisi üzerinde en performanslı şekilde çalışabilmesini olanak sağlamaktadır. Belirtilen konseptlerin tercihi ve uygulama şekilleri ile ilgili örnek senaryolar aşağıda bulunmaktadır.

Windows Azure Blob Storage, Page Blob

İş uygulamaları, içerisinde bulunduğu sürece bağlı olarak, yoğun veri okuma süreçlerin de bulunmaktadır. Örneğin; Yoğun video yayının yapan iş uygulamaları, istemcilerini besleyebilmek için talep edilen video içerikleri yoğun okuma işlemlerini gerçekleştirmesi gerekmektedir. Söz konusun süreç de yoğun okumalara en iyi performans yakalayabilmesi amacı ile **Windows Azure Page Blob** konseptinin tercih edilmesi daha sağlıklı sistem altyapısı oluşturmanıza olanak sağlayacaktır.

Windows Azure Blob Storage, Block Blob

İş gereksinimleri zaman içerisinde paralel olarak, yoğun dosya yükleme işlemleri gerektirebilmektedir. Örneğin; E-ticaret sistemleri ürün içeriklerini güncellemek amacı ile günlük içerik düzenleme süreçleri gerçekleştirmektedir. Gerçekleştirilen işlemler ile birçok ürün fotoğraf içerikleri yüklenmektedir. İzlenen sürecin en performanslı olarak gerçekleştirmek amacı ile **Windows Azure Block Blob** konseptinin tercih edilmesi başarılı sonuçlar alınmasını sağlayacaktır.

*Not: Yapılan anlatımlarda "**WindowsAzure.FunnyApp**" uygulama örneği kullanılmıştır. Uygulama ile ilgili kaynak kodları aşağıdaki bağlantıyı kullanarak edinebilirsiniz.*

Github / https://github.com/ibrahimatay/WindowsAzure.FunnyApp

İş uygulamaların başarılı ve performanslı olarak çalışabilmesi için geliştiricilerin, iş tecrübelerine göre çeşitli yaklaşımlar uygulanmaktadır. Günümüzde depolama ve verinin kullanımı konusunda yeni konseptler oluşturulmaktadır. Microsoft **Windows Azure Platform** ile de sürece yeni bir bakış açsı oluşturulmuştur. Konu ile ilgili sorularınızı info@ibrahimatay.org eposta adresine yöneltebilirsiniz.

İbrahim ATAY

Understanding Block Blobs and Page Blobs

http://msdn.microsoft.com/en-us/library/ee691964.aspx

RPC 2616

http://www.ietf.org/rfc/rfc2616.txt