Bluezone – Innovation Project

Solution overview

blue-zone AG wanted to lift up their cloud-based product for sales and technical field service to a modern cloud-based environment to be able to offer their products as a Cloud Solutions Provider (CSP). In a joint effort Microsoft Deutschland GmbH and blue-zone AG migrated their existing solution into the App Service world.

Key technologies used

- Azure App Services
- Azure WebJobs
- SQL Azure

Core team

- Richard Mayr | CEO - Stakeholder
- Matthew Witherspoon | Developer
- Dariusz Parys | Technical Evangelist

Customer profile

Since its founding in 2011, blue-zone AG has specialized on the development of digital sales and service systems for cloud & mobile computing. Its range of services spans from standardized turnkey SaaS solutions to highly customized implementations with full system integration. R&D is an integral part of blue-zone's DNA. The company's internal expectations always start with the implementation of mature, innovative solutions that harness the full business potential of mobile computing, not least by making tablet usage (iOS and Windows) economic and adding a demonstrable competitive edge to all users. The focus is on blue-app, a mobile field force automation solution to boost the success of sales and service teams by equipping them with a full range of information and functions, whether online or offline. blue-app today provides manufacturers and retailers with a powerful basis for accelerating their sales process, and for boosting cost efficiency and precision. As a certified Microsoft partner, blue-zone offers the mobile Field Force Automation app in conjunction with Microsoft's secure, powerful and highly scalable Azure Cloud. blue-zone also supports the use of blue-app in individually defined (private) cloud environments or internal (on premise) infrastructures to promote seamless integration of CRM, ERP and PIM systems.

Website

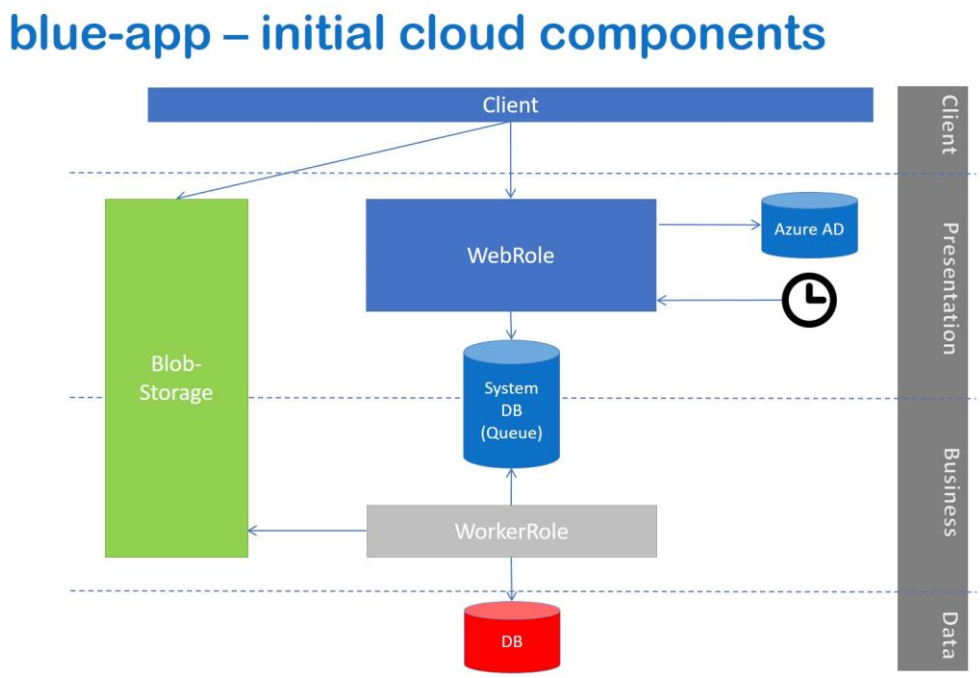
Problem statement

blue-zone AG's cloud based products **blue-app** and **blue-next** have been developed on the classic Azure model with cloud-based worker and web roles. Due to an important change in their business model there was the need to migrate those apps into a modern architecture leveraging Azure App Services and replacing the existing cloud-services.

Solution, steps, and delivery

Starting Position

blue-app's current architecture has the following components as shown in the following architectural diagram

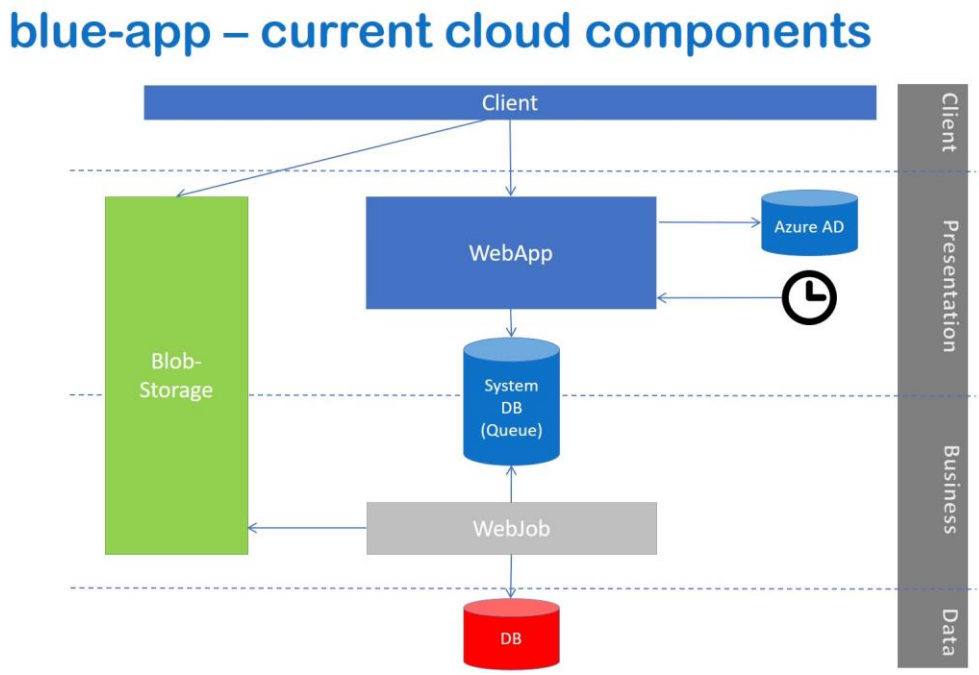


Basically the clients connect to the application through *Active Directory* to the `WebRole` and schedule tasks to retrieve new data for their mobile devices through a queuing system which is based on *SQL Azure* . The `WorkerRole` picks up the work and creates artifacts that are stored in the *Blob Storage*. The mobile clients are picking up the created artifacts which contains data to be used by the field.

The two components that have to be migrated are the `WebRole` and `WorkerRole`. One of the main goals of the migration effort was to reuse as much code as possible with the least amount of time. This mantra had some impact on technical decisions.

Migrating the WebRole to App Services

blue-app's rewritten architecture changed in the two areas as shown below



The replacement of the `WebRole` to the *Azure App Service Web App* was pretty straightforward. As the logic behaves the same the code could be adopted nearly line by line to the new model with respect to the same queuing mechanism used as in the version before. Here is an example of creating a work item for a client request

```
//blue-app

// methods behind client facing web service
// receives client data version and if necessary, creates the work item
public updateResponse_2 CreateCustomerWorkItem()
{
         if (!IsComputationNecessary())
         {
                  return CreateEmptyUpdateResponse();
    }

    if (SystemDatabaseService.IsRequestBlocked("GetCustomerData",
connectionStringProvider))
    {
                  log.Warn("CreateCustomerWorkItemUsingRequest2: GetCustomerData requests
are blocked");
                  return CreateEmptyUpdateResponse();
         }

    var response = new updateResponse_2();
    response = IsInitial() ? CreateInitialCustomerResponse(response) :
CreateUpdateCustomerResponse(response);
    response.maxVersion = HighestValidVersion.ToString(CultureInfo.InvariantCulture);
    return response;
}

//checks the existence of the work item and creates a new work item one does not already
exist
private Guid CreateInitialCustomerWorkItem()
{
         Guid initCustWiGuid = Guid.Empty;
         bool initCustWiExists =
WorkItemService.HasCustomerWorkItemBeenCreated(connectionStringProvider, LastVersion,

HighestValidVersion, userName, out initCustWiGuid);
```

```
    return initCustWiExists ? initCustWiGuid :
WorkItemService.CreateInitialCustomerWorkItem(connectionStringProvider, ShardId, userName,
HighestValidVersion, ExporterType);
}
```

Migrating the WorkerRole to Azure Web Jobs

The second cloud-based service the `WorkerRole` was migrated to *Azure WebJobs*. The decision to go with *Azure WebJobs* instead of *Azure Functions* was just based on the fact that the migration has to be going fast and the team already had experience with *Azure WebJobs*.

The following snippet shows the code for the *Azure WebJob*. Previously, each worker role had 10+ threads running this method and competing for async jobs. Now each WebJob has this method and competes for the async jobs.

```
public void Run()
{
    while (!stop)
    {
                WorkItems wi = null;
                CurrentWorkItemGuid = null;
                try
        {
                            wi = WorkItemService.GetNextWorkItem(connectionStringProvider,
workItemConcurrencyLimits);

            if (wi.Discriminator == WorkItemType.nothingToDo.ToString())
            {
                                //nothing to do, sleep for 1 to 5 seconds
                Thread.Sleep(r.Next(1000, 5000));
                continue;
            }

            log.Info("Run: workItem guid {0}, status {1}, type {2}", wi.Guid, wi.Status,
wi.Discriminator);

                            if (wi is CustomerWi)
            {
                var customerWi = (CustomerWi)wi;

                                HandleCustomerWorkItem(customerWi);
            }

                            if (wi is CatalogWi)
            {
                                var catalogWi = (CatalogWi)wi;
                HandleCatalogWorkItem(catalogWi, catalogWi.IsInitial ? new
InitialCatalogExporter(connectionStringProvider) : new
UpdateCatalogExporter(connectionStringProvider));
            }

                            if (wi is AssetWi)
            {
                                var assetWi = (AssetWi)wi;
                HandleAssetWorkItem(assetWi, wi);
            }

                  }

                  WorkItemService.UpdateStatus(connectionStringProvider, wi.Guid,
WorkItemStatusEnum.FinishedSuccessfully);

                catch (Exception e)
        {
                            var exceptionString =
ExceptionHelper.GetCompleteExceptionTreeAsString(e);
```

```
            if (wi != null)
            {
                                    WorkItemService.RecordError(connectionStringProvider,
wi, exceptionString);
                LocalFileFacade.DeleteLocalFile(connectionStringProvider, wi.Guid);
                log.Error("Run: exception thrown while running work item {0} of type {1}
{2}", wi.Guid, wi.Discriminator, exceptionString);
            }
            else
            {
                                    log.Error("Run() an exception was thrown before the
work item was loaded {0}",
                            exceptionString);
            }
        }
         }
}

//method used to prioritize the work items which are ready for processing
//only one thread or webjob is able to enter this method at a time
internal static WorkItems GetNextWorkItem(SystemControlEntities sysContext,
Dictionary<WorkItemType, int> workItemConcurrencyLimits)
{
         const int working = (int)WorkItemStatusEnum.Working;
         WorkItems wi = new CatalogWi { Discriminator = WorkItemType.nothingToDo.ToString()
};

    WorkItemTuple[] eligibleWorkItems = LoadEligibleWorkItems(sysContext);
    if (!eligibleWorkItems.Any()) return wi;

    WorkItemTuple[] runningWorkItems = LoadRunningWorkItems(sysContext, working);

    foreach (var workItem in eligibleWorkItems)
    {
                if (AreTooManyWorkItemsOfThisTypeRunning(runningWorkItems,
workItem.Discriminator, workItemConcurrencyLimits, workItem.ShardNumber)) continue;

        if (IsDelayNecessaryDueToFailureInLastAttempt(workItem)) continue;

        if (!IsPredecessorFinished(sysContext, workItem.Predecessor)) continue;

        if (!IsPredecessorFinished(sysContext, workItem.Predecessor2)) continue;

        if (IsWorkItemScheduledForFutureDate(sysContext, workItem.Discriminator,
workItem.Guid)) continue;

        wi = LoadCorrespondingChild(sysContext, workItem.Discriminator, workItem.Guid);

        wi.Attempts++;
        wi.Status = working;
        wi.FirstAttemptStartTime = wi.FirstAttemptStartTime ?? DateTime.UtcNow;
        wi.AttemptStartTime = DateTime.UtcNow;
        break;
    }
    return wi;
}
```

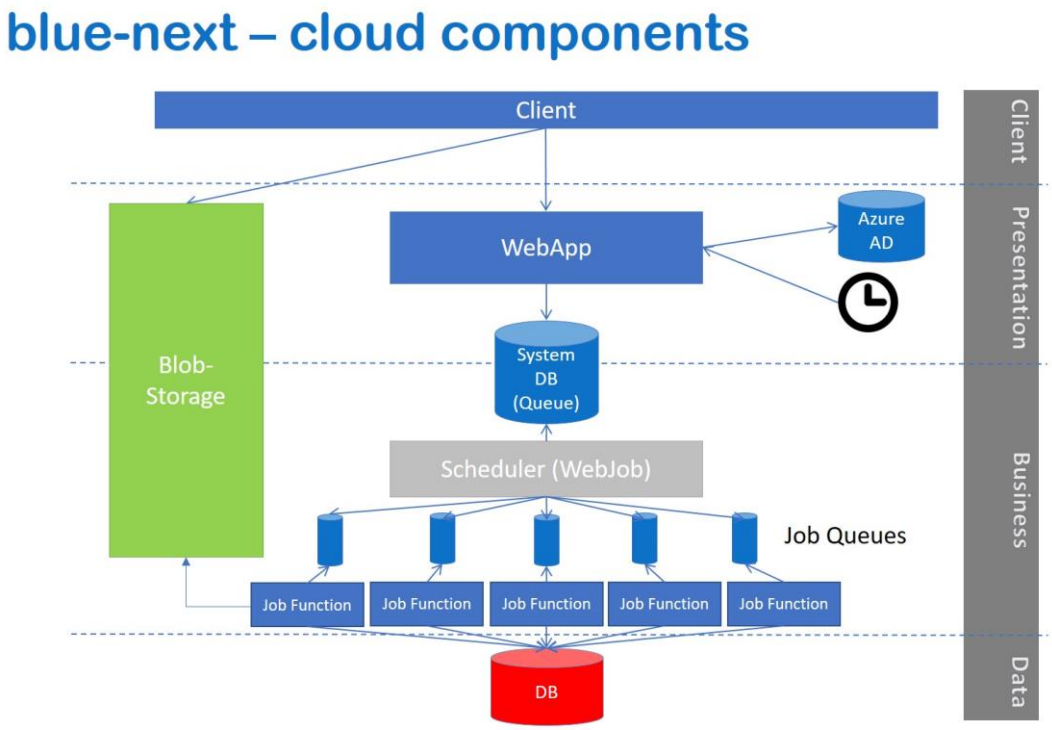
Conclusion

There are several commercial benefits using the new architecture over the classic model used before. Those are

- Reduction in idle resources
    - Async *Azure WebJobs* use the pool of resources available to the *Web App*
    - *Web Apps* may be scaled out based on additional parameters
- Faster deployments

    - Deployments no longer tear down and set up Roles like it was before.
    - *Azure WebJobs* are deployed seperately
- Resources created using ARM templates
    - One of the bigger benefits is the use of Azure Resource Manager Templates short ARM which significantly reduces project start time
    - Standardization of environment settings and be able to redeploy the same infrastructure with ease.
- Improved monitoring out of the box compared to the classic model.

The next version called **blue-next** is already in development and will enhance the architectural blueprint with some new addditions



like *Azure Functions* and *Service Bus Queues* to leverage more distributed computing power.