Partnering with CAST to extend and migrate CAST Highlight from AWS to Microsoft Azure

Stéphane Goudeau

CAST Highlight is a recent offering from CAST Software that enables companies to quickly and objectively measure the risk, complexity, and cost of their application portfolios. The solution provides exclusive insights into application risks and opportunities before any investment, rationalization, or retirement decision is made on an IT asset.

A joint team from CAST and Microsoft worked to define rules that assess the ability of an existing codebase to migrate to Microsoft Azure. The team then integrated the rules into CAST Highlight and moved the solution itself to Azure.

In this report, we describe the process and what we did before, during, and after the hackfest, including the following:

- How we produced the rules that assess the ability to migrate to Azure
- How we benchmarked the rules
- How we migrated the CAST Highlight service to Azure
- What the architecture looked like and future plans
- Learnings from the process

CAST Overview

CAST is an independent software vendor that is a pioneer and world leader in software analysis and measurement (SAM). With more than $120 million cumulative investment in research and development, CAST provides the most advanced technology in the world to capture and quantify the reliability, security, complexity, and size of business applications. CAST introduces fact-based transparency into software asset management (including application development, maintenance, and sourcing) to transform it into a management discipline. More than 250 companies across all industry sectors and geographies rely on CAST to prevent business disruption and build a transparent dialogue with their service providers. CAST is also an integral part of software delivery and maintenance at the world's leading IT service providers and independent software vendors.

Problem statement

Our first objective was to define rules that assess the ability of applications to migrate to Azure and integrate those rules into CAST Highlight. This was the more-complex task for our team.

Our second objective was to move the existing application to Azure, thus profiting from App Service features such as auto-scaling and deployment slots. The existing application is a Java web app running on Apache Tomcat and using PostgreSQL as its database. This is a frequent scenario for web applications running in Azure, so we did not anticipate having any issues with this task.

Solution, steps, and delivery

The key questions that need to be answered are

- Which applications could be moved?
- How could they be moved?
- How to prioritize them?

To understand which applications should be moved (and when and how), it's important to create a well-attributed catalog of these applications. The relative importance of each attribute can then be weighted and the prioritized list can be built.

Analyzing the applications from a bottom-up perspective is aimed at providing a view into the eligibility, at a technical level, of an application to migrate. This should be based on both business and technical questions and on code scanning.

Depending on the existing on-premises application, several Azure services may be considered. CAST wanted to propose for each service a technical analysis based on the existing code that would enable evaluation of the level of complexity of migration for each of the Azure targeted services. The final result of the assessment would be called the "CloudReady index." With this value, any company can figure out the applications that have the highest potential value and that are best suited for migration.

Business questions and technical questions would both be handled through an e-form quick interview. Technical questions would be answered automatically via a pure code analysis.

Business Considerations
Cost Effectiveness
Data Considerations
Service Level Agreement (SLA) Requirements
Latency
Provisioning
Migration Costs
Geopolitical Concerns

Technical Considerations
Communications & Messages
Data (Nature, Access & Storage)
Security, Identity & Access Control Management
Technologies/3rd Party Integrations
Application Lifecycle Management (ALM)
Infrastructure/Deployment/Setup/Versioning
Operations (monitoring, backup & recovery, change management (see above), governance, etc.)

The automated rules present the following attributes:

- Dependent on a decision tree.
    - Analysis is to be executed on a specific targeted service; for example, the Web Apps feature of Azure App Service.

    - Analysis can be launched simultaneously on several services, such as Web Apps and Service Fabric.
    - Depending on the targeted service, rules can vary.
- Weighting of answer for a given rule.
    - Unsupported feature corresponds to a negative score.
    - Supported feature corresponds to a bonus.
- Rules classification: pattern, language, framework, technology, and so on.
- Complexity: low, moderate, high.
- Impact: code, framework, architecture. We consider the architecture to be impacted as soon we add a new Azure component (even Azure Storage).

Code analysis is based on identification of specific patterns, such as the following:

- Persistent files
- Temporary files
- App settings configuration
- Registry settings
- Shared caching
- Application logs
- Sensitive data storage protection
- Data encryption keys
- Users authentication
- Code execution
- Services and scheduled tasks
- Inter-applications messaging

The presence or absence of a pattern may be interpreted as a bonus or a negative score, with a specific weight depending on the impact on code, framework, or architecture. Each pattern is documented as in the following sample:

Services and scheduled tasks	
Technical Gap	Windows services or scheduled tasks cannot be deployed on Azure Web Apps.
Migration path	Corresponding feature has to be implemented as continuous and scheduled WebJobs or Azure functions
Identified tasks	Detect System.ServiceProcess, HostingEnvironment.QueueBackgroundWorkItem, Quartz.CronScheduleBuilder Transform and adapt the existing service or scheduled task code to the required interfaces
Impact	Code, Architecture
Complexity	Moderate/High
Reference	https://msdn.microsoft.com/en-us/library/d56de412(v=vs.110).aspx http://apidoc.quartz-scheduler.net/ https://msdn.microsoft.com/en-us/library/dn636893(v=vs.110).aspx https://azure.microsoft.com/en-us/documentation/articles/web-sites-create-web-jobs https://azure.microsoft.com/en-us/documentation/articles/functions-overview

Moving CAST Highlight to Azure

We divided the process of migration into the following steps:

1. Analyze existing architecture and technical requirements
2. Propose the first version of the architecture
3. Start to migrate the application
4. Gather the experience
5. Propose the second version of the architecture
6. Iterate

The original architecture consisted of the following:

- A cloud storage subsystem with programmatic CRUD capabilities
- A Java web application running on a Tomcat server
- A PostgreSQL database running on a Linux server

We wanted to leverage the platform as a service (PaaS) architecture and reuse the existing database.

Front end

The front end of CAST Highlight uses Java technologies. The only concern that CAST expressed was the level of Java support in Azure. That concern was eliminated after looking at a Java website that uses the customizable installation of Apache Tomcat for Web Apps. (Apache Tomcat is an open-source software implementation of the Java servlet and Java Server Pages technologies for Azure App Service.)

The next step was simple. We just had to verify that the default web configuration was OK for this Java application before publishing the app.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <handlers>
            <add name="httpPlatformHandler" path="*" verb="*"
                 modules="httpPlatformHandler" resourceType="Unspecified"
/>
        </handlers>

        <httpPlatform processPath="d:\home\site\wwwroot\bin\apache-tomcat-
8.0.33\bin\startup.bat" arguments="start">
            <environmentVariables>
                <environmentVariable name="CATALINA_OPTS" value="-
Dport.http=%HTTP_PLATFORM_PORT%" />
                <environmentVariable name="CATALINA_HOME"
value="d:\home\site\wwwroot\bin\apache-tomcat-8.0.33" />
                <environmentVariable name="JAVA_OPTS"       value="-
Djava.net.preferIPv4Stack=true" />
            </environmentVariables>
        </httpPlatform>
    </system.webServer>
</configuration>
```

The CAST Highlight website was then deployed to Web Apps with FTP.

Connectivity to the backend

We used Azure Virtual Network integration to enable access from the web app to a database running on a virtual machine (VM) in an Azure virtual network. With Virtual Network integration, you don't need to expose a public endpoint for applications on your VM but can use the private non-Internet routable addresses instead.

Because we decided to use Resource Manager, it was quite easy to create a new virtual network configured with a gateway and point-to-site connection: In the networking UI of App Service, simply select "Create New Virtual Network."

Database layer

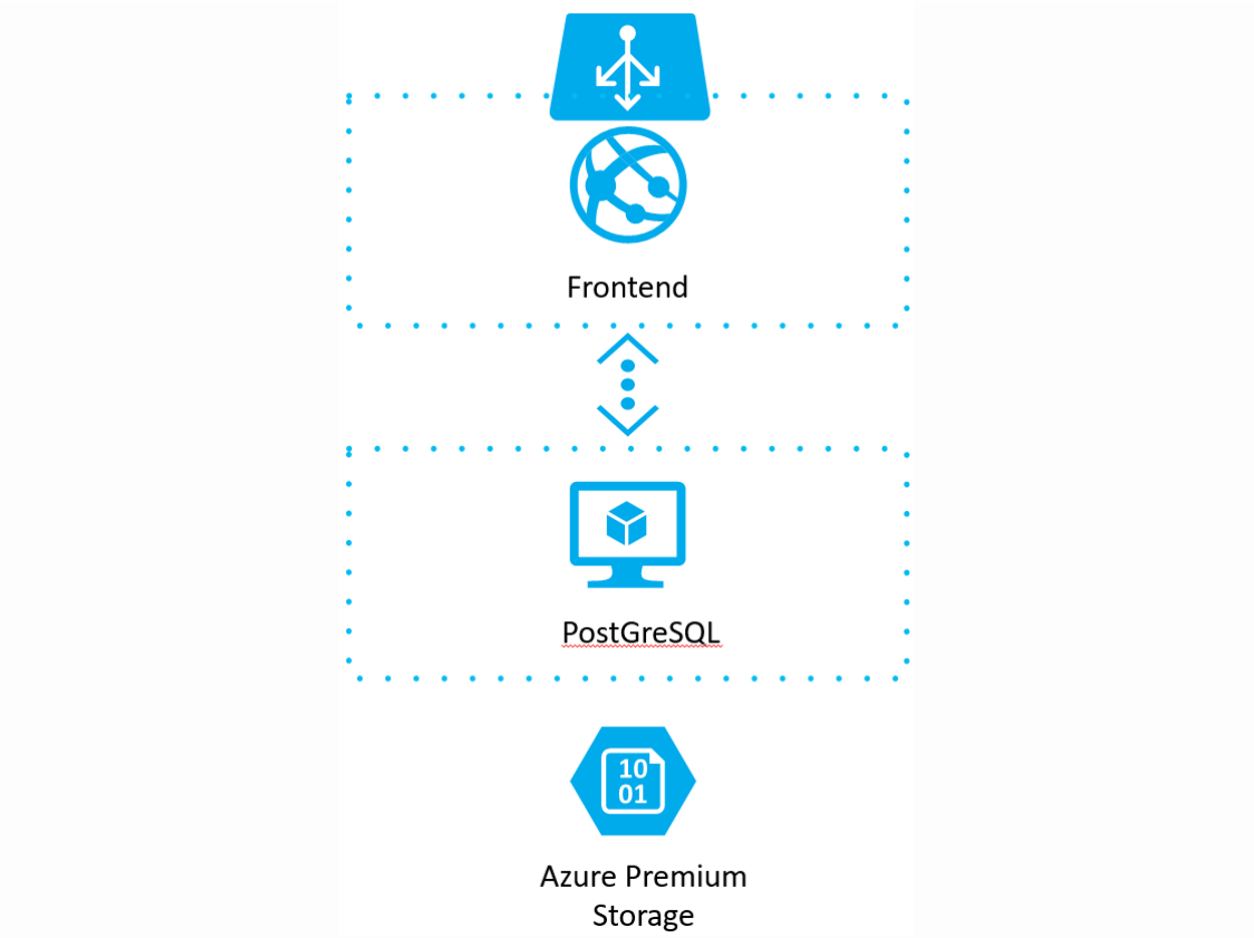
Because a production release of "PostgreSQL as a Service" doesn't yet exist in Azure, the only option was to install and set up PostgreSQL on the Linux VM deployed on the integrated virtual network to

which we had connected the Azure web app. The database was then installed on this PostgreSQL instance.

Storage

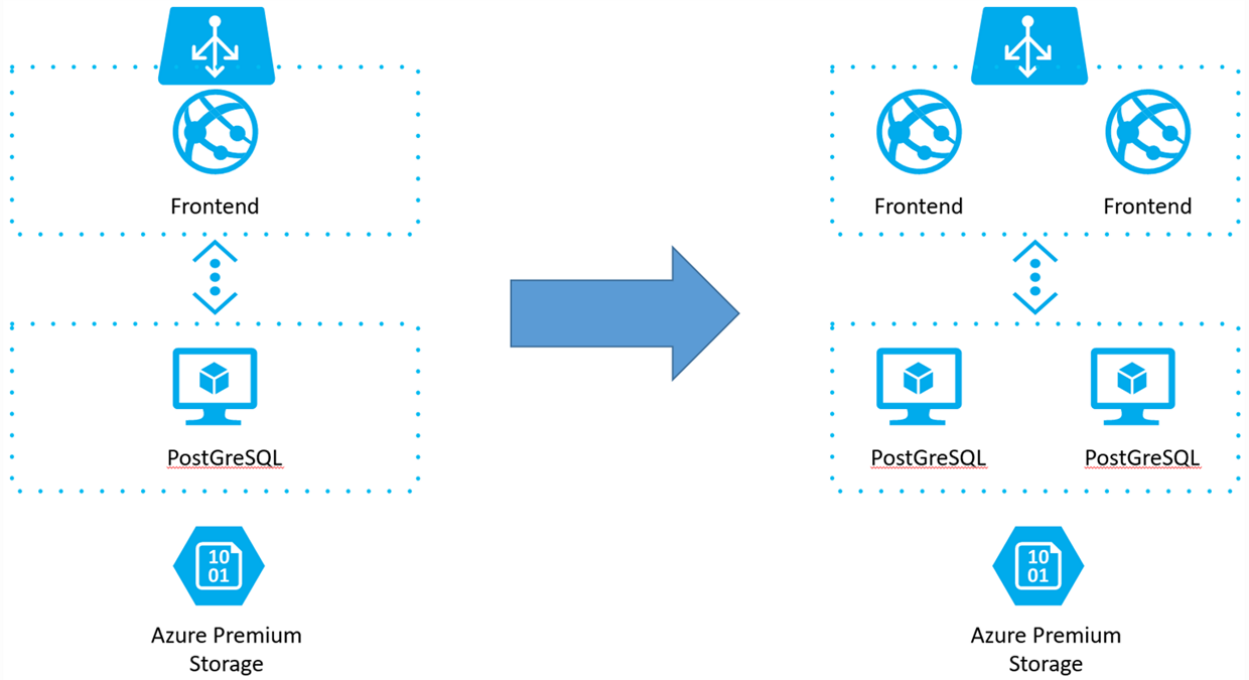
We decided to use Azure Premium Storage. With Premium Storage, you can attach multiple persistent data disks to a VM and configure them to meet your performance and latency requirements. Each data disk is backed by an SSD disk for maximum input/output performance. This choice also enables CAST to take advantage of the Microsoft single-instance SLA of 99.9 percent uptime.

At the end of this iteration, we had a very simple architecture, as shown in the following figure:



In a future iteration, the PostgreSQL database might be deployed to a high-availability cluster on Azure Virtual Machines so that the CAST Highlight solution could guarantee business continuity of service.

The future architecture should evolve in the following way:



Migration steps

We first created a web app from the default “Apache Tomcat 8” template. This approach proved to be useful because this environment corresponded almost exactly to the target. We used FTP to publish directly to the web app. The Catalina config file was modified to import three CAST-specific libraries that were not in the default environment. We established a connection to an integrated virtual network through the UI. And we cloned an existing Linux + PostgreSQL VM back end in this virtual network (by using AzCopy to perform a VHD blob copy and a PowerShell script to integrate it into the newly created VM).

We did not encounter any specific issues during the App Service migration. Using Kudu with the default Apache Tomcat deployment helped a lot to identify what was required to launch the application. (It took less than 30 minutes for this part.)

“The solution to publish a Java web app on Azure is really simple” was the shared feedback about the usage of Azure App Service in our context.

Conclusion

During the technical engagement between Microsoft and CAST, we leveraged skills from both companies in defining the CloudReady index rules and in migrating the existing Java + PostgreSQL app to a mixed IaaS-PaaS solution based on Azure App Service.

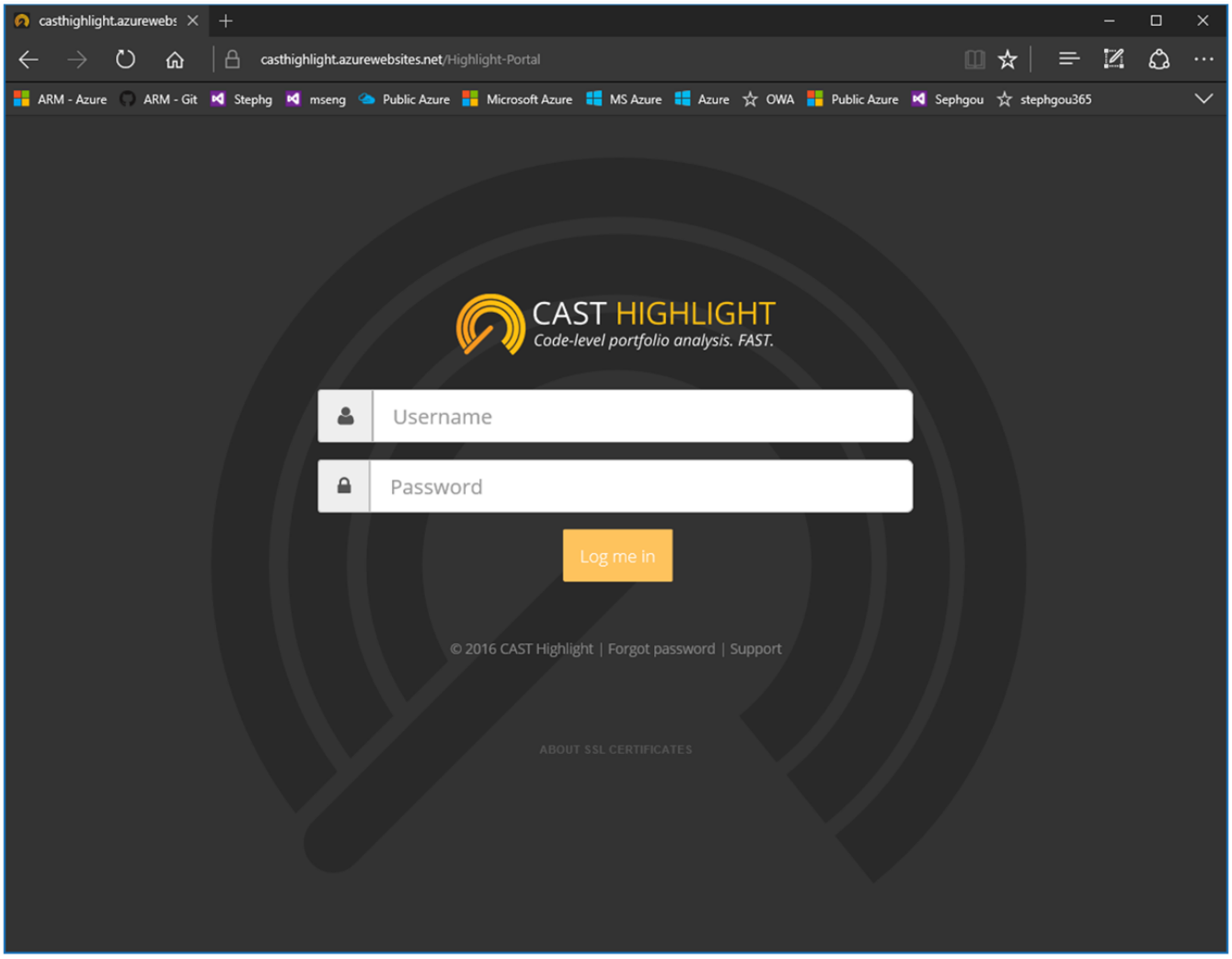
Since then, we have completed a pilot project with a few ISVs, and the results have been immensely insightful. This enabled us to adapt the various weights related to questions and patterns.

In the near future, with the anticipated business success of the CAST Highlight service, the architecture will leverage the autoscaling PaaS benefits and will evolve to eliminate the chance of any availability issues on the back end.

The CAST team spoke positively not only about how the Azure cloud platform works but about how the Microsoft platform works with the open source.

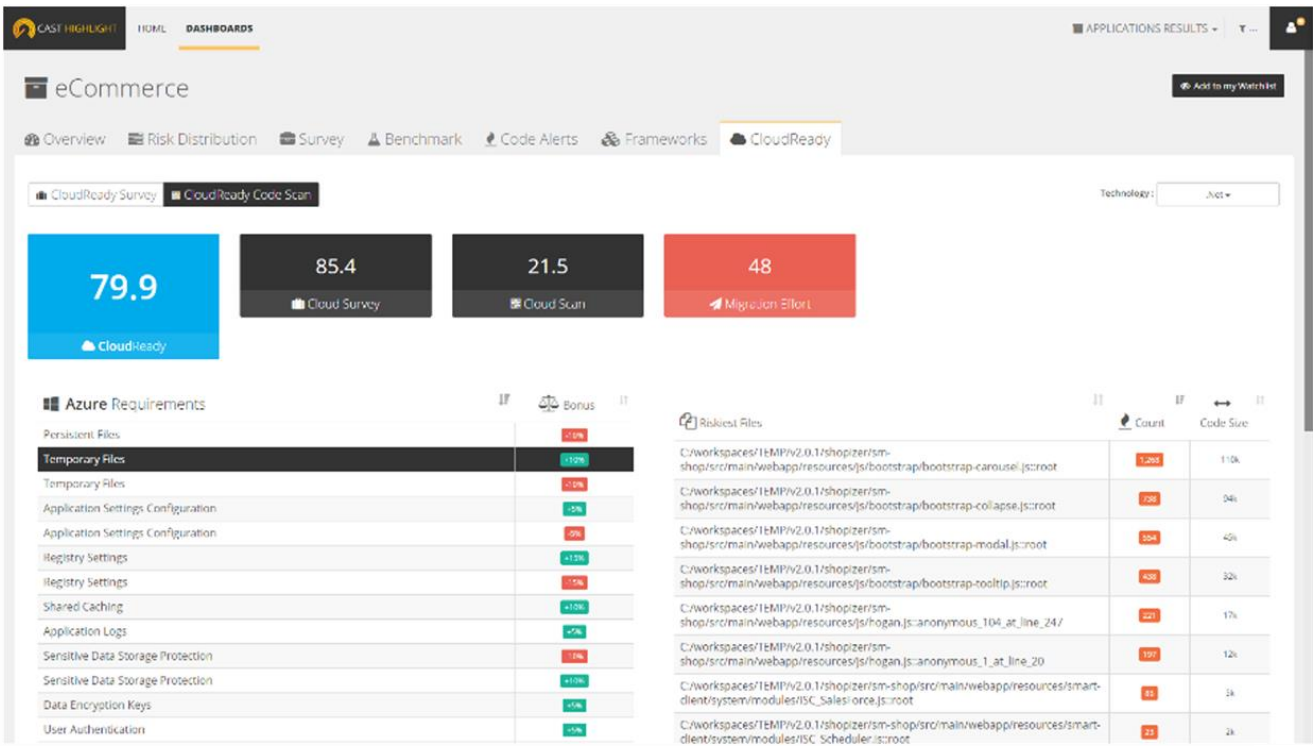
The CAST system is written in Java and uses PostgreSQL, so for successful migration it was necessary to ensure that the technologies used are supported and tested in the cloud. Both Java and PostgreSQL can be described as “first-class citizens” on Azure, and it was one of the important learnings for the team.

The CAST Highlight solution is now available as a service deployed in Microsoft Azure.



The resulting CloudReady-ness assessment from CAST Highlight enables customers to classify and prioritize applications before their migration to the cloud.

The code analysis contains detailed key information on what needs to be changed.



CAST Highlight should be of great help for any company that needs to migrate its apps portfolio to the cloud.