**Cast Highlight:**

Cast Highlight gives you the insights to increase your Software IQ.

It mainly focuses on App Portfolio Analysis, Cloud Readiness, Agile Analytics and Software Engineering.

They care about data privacy. At any moment, your source code doesn’t leave your infrastructure. Scan your application locally with our Highlight Agent, then upload metrics.

Currently supports Java, C# and VB.Net.

Export results to CSV, XML and PPT

How to Use?  
Download the application locally.

Run the app and get a csv file as output which is encrypted.

Upload the csv file and get back the report.

**Comparison between Cast Highlight and VSM:**

**Topics which are covered by Cast Highlight or VSM:**

|  |  |  |
| --- | --- | --- |
|  | **CAST HIGHLIGHT** | **VSM** |
| User Authentication | Yes | Yes |
| Application Logs | Yes | Yes |
| Persistent Files | Yes | Yes |
| Temporary Files | Yes | Yes |
| Application Settings Configuration | Yes | Yes |
| Registry Settings | Yes | Yes |
| Access Control List | Yes | No |
| Code Execution | Yes | No |
| Data Encryption Keys | Yes | No |
| Inter-Application Messaging | Yes | Yes |
| Sensitive Data Storage Protection | Yes | Yes |
| Services & Scheduled Tasks | Yes | Yes |
| Shared Caching | Yes | Yes |
| Third-Party External Dependencies | Yes | Yes |

**Features:**

|  |  |  |
| --- | --- | --- |
|  | **CAST HIGHLIGHT** | **VSM** |
| User Authentication | Yes | Yes |
| Graphical Representation of the generated report | High | Low |
| Cloud Ready Points | Yes | Yes |
| Cloud Ready Survey | Yes | No |
| Cloud Ready Scan | Yes | Yes |
| Boosters/Suggestions | Yes | Yes |
| Blockers/Antipatterns | Yes | Yes |
| Roadblocks/Warning | Yes | Yes |
| Software Maintenance | Available in FTE | Available as complexity points |

**Specific Patterns / Rules:**

|  |  |  |
| --- | --- | --- |
|  | **CAST HIGHLIGHT** | **VSM** |
| Access Control List | Yes | Yes |
| Application Settings Configuration - Using other configuration files than Web configuration | Yes | Yes |
| Execution Environment - Application server dependencies | Yes | Yes |
| Using File System | Yes | Yes |
| Using Hardcoded IP Addresses | Yes | Yes |
| Using a middleware application | Yes | Yes |
| Persistent Files - Perform File Manipulation | Yes | Yes |
| Persistent Files - Using stateful session (Servlet) | Yes.  Using,  getSession().setAttribute( | Yes.  Using,  getSession().getAttribute( |
| Security & User Authentication - Using impersonate Identity | Yes | No |
| Security & User Authentication - Using of unsecure network protocols (HTTP, FTP) | Yes | No |
| Temporary Files - Access to environment variable | Yes | No |
| Application Server Dependencies | Yes | Yes |
| Perform Directory Manipulation | Yes | No |
| Using Stateful Sessions (Spring) | Yes | No |
| Using Stateful Sessions (Javax Servlet) | Yes | Yes |

**Cast Highlight uses the following patterns for code analysis:**

1. Access Control List - Using Access Control List:

Managing user rights based on ACL assumes that the application is running in the identity context of the user, which is not available in a Cloud environment. As a migration task, you should identify within your application the usage of System.Security.AccessControl related functions as no access security list is available on the Cloud. New implementation will be defined on case-by-case basis.

**Pattern:**

**Java:** look in source code for usage of import java.nio.file.attribute or implements AclFileAttributeView

**C#/VB/VB.Net:** look in source code for Using System.Security.AccessControl and (File.SetAccessControl or File.GetAccessControl)

1. Application Settings Configuration - Using other configuration files than Web configuration:

Storing application settings in external files other than web.config and that are not manageable in the Cloud platform is not recommended as such settings will not be easily changeable.

**Pattern:** Look in project folders for \*.config different than web.config

1. Execution Environment - Application server dependencies**:**

**Pattern:**

**Java:** import org.eclipse.jetty, import weblogic

1. Using File System:

Cloud applications should not assume the local file system is accessible, as the directory structure might be different from a traditional desktop or server machine and/or the Cloud application may not have sufficient rights to access the local file system. Instead, use relative paths to application resources (e.g. ../../reporting/reportBuilder.xml). Depending on your application context and the Cloud platform where it is deployed, you could also consider using functions or classes like LocalResources to dynamically resolve file paths.

**Pattern:**

C:\, D:\ … Z:\ for Windows platforms

/var, /user, /etc for Linux platforms

1. Using Hardcoded IP Addresses:

From a software engineering standpoint, and especially in the Cloud, applications should be developed like we don’t know what is the underlying infrastructure. Using, calling or referencing remote resources by using hardcoded IP addresses should be avoided as they can regularly change. Instead, it is recommended using domain names or eventually store IP addresses in a configuration file. This CloudReady pattern verifies the presence of IPV4 or IPV6 addresses in the source code.

**Pattern:** Look in source code URLs corresponding to IPV4 (e.g. 216.58.213.174) and IPV6 (e.g. 2001:0db8:85a3:0000:0000:8a2e:0370:7334) structures.

1. Using a middleware application:

Existing application may use asynchronous messaging middleware that enables to send data between decoupled systems. Most of the time, these messaging environments are not natively integrated in PaaS services and should be replaced by Cloud-based solutions. Identifying upfront and at the portfolio level the applications using such components is key to anticipate, plan and optimize technical tasks of the migration.

**Pattern:**

**Java:** import ionic.Msmq, com.rabbitmq, com.tibco.tibjms, com.ibm.mq.jms  
**C#/VB/VB.Net:** Using System.Messaging, RabbitMQ.Client, TIBCO.EMS, IBM.WMQ

1. Persistent Files - Perform File Manipulation :

Manipulating local files requires specific permissions and usually assumes the file will be persisted over time. In the Cloud, because the underlying infrastructure can be moved or removed, it is not possible to make such assumptions. Instead of using the file system, store your temporary information in a dedicated Cloud-based storage or in a NoSQL database.

**Pattern:**

**Java:** import org.apache.commons.io.FileUtils; or import java.io.File; and moveFile()or forceDelete() or deleteQuitely() or copyFile() or write() …

**.Net:** Using System.IO and File.Move() or File.Create() or File.Copy() or File.Exists() or File.Encrypt() orFile.Decrypt() or File.Open() …  
Except when code using File.\* contains Path.GetTempPath()

1. Persistent Files - Using stateful session (Servlet):

For modern applications running in the Cloud, it is not recommended to be stateful, especially for sessions as they’re not scalable, and are generally harder to replicate and fix bugs (server-side). Ideally, stateful sessions should be replaced by stateless and client-side mechanisms such as cookies, client cache (e.g. Redis, memcache…) or in an external cloud-based storage. This is an important architectural constraint of microservices-style applications, as it enables resiliency, elasticity, and allows any available service instance to execute any task.

**Pattern:**

Old Rule:  
Looking for .setAttribute(

import javax.servlet.http.HttpSession; and getSession().setAttribute( or getSession().putValue(

1. Persistent Files - Using stateful session (Spring):

For modern applications running in the Cloud, it is not recommended to be stateful, especially for sessions as they’re not scalable, and are generally harder to replicate and fix bugs (server-side). Ideally, stateful sessions should be replaced by stateless and client-side mechanisms such as cookies, client cache (e.g. Redis, memcache…) or in an external cloud-based storage. This is an important architectural constraint of microservices-style applications, as it enables resiliency, elasticity, and allows any available service instance to execute any task.

**Pattern:**

Look in “WEB-INF/applicationContext.xml” for  
<beans>  
<bean id=”accountService” class=”com.foo.DefaultAccountService” scope=”session”/>  
…  
</beans>

1. Security & User Authentication - Using impersonate Identity:

The primary reason for impersonation is to cause access checks to be performed against the client’s identity. Using the client’s identity for access checks can cause access to be either restricted or expanded, depending on what the client has permission to do. For example, suppose a file server has files containing confidential information, and each of these files is protected by a [DACL](https://msdn.microsoft.com/en-us/library/cc246052.aspx). To prevent a client from obtaining unauthorized access to information in these files, the service can impersonate the client before accessing the files.

Some on-premise applications may rely on executing code with the identity of the actual Windows user. This requires a corresponding user account has been created on the platform, which is not possible in a Cloud environment. User authentication should be replaced by a [Cloud-based and OS-agnostic identity & access management solution](https://solutionsreview.com/identity-management/identity-management-solutions-directory/) such as Active Directory.

**Pattern:**

**Java:** Look in source code for import waffle.windows.auth.\*

**.Net:** Look in web.config for <identity impersonate=”true”  
 Look in source code for impersonationOption.\*

1. Security & User Authentication - Using of unsecure network protocols (HTTP, FTP):  
   Using secured protocols such as HTTPS and SFTP (over HTTP and FTP) should now be the norm as applications are more and more exposed and interconnected. This CloudReady patterns looks for unescured URI in the source code. Ideally, URLs should be replaced in your source code by secured protocols HTTPS and SFTP (and ensure the infrastructure implements these protocols for the resources your application calls, uses or references).

**Pattern:** Look in source code URL preceded by http:// or ftp://

1. Temporary Files - Access to environment variable:

Environment variables are OS-dependent and as such, not Cloud-friendly. Additionally, their existence in a Cloud environment cannot be guaranteed over time as the underlying infrastructure could be moved, remove or duplicated. Instead, you should consider using YAML files or shared data source to store your application configuration parameters.

**Pattern:**

**Java:** Look in source code for import java.lang.\*; and System.getenv()

**.Net:** Look in source code for Using System.Environment.GetEnvironmentVariable()  
or Using Environment.ExpandEnvironmentVariables()

1. Application Server Dependencies:

Generally speaking, Cloud-based App services run in a secure environment called a sandbox. The sandbox generally aims to restrict access to shared components of Windows. Each app runs inside its own sandbox, isolating its execution from other instances on the same machine as well as providing an additional degree of security and privacy which would otherwise not be available. Unfortunately, graphics subsystems of Windows or Global Assembly Cache have been designed as shared components.

**Pattern:**

**Java:** import org.eclipse.jetty, import weblogic

1. Perform Directory Manipulation:

Manipulating local directories requires specific permissions and usually assumes a predefined directory structure exists. In the Cloud, it is not possible to make such assumptions. If your application requires to create, delete or modify folders, implementing IAM (Identity & Access Management) solutions should be considered.

**Pattern:**

**Java:** import org.apache.commons.io.FileUtils; and moveDirectory() or MoveDirectoryToDirectory() or deleteDirectory() or copyDirectory() or copyDirectory() or copyDirectoryToDirectory()…

import java.io.File; and renameTo() or mkdir() or mkdirs() or isDirectory() or delete()

.Net : Using System.IO and Directory.CreateDirectory() or Directory.Move() or Directory.Delete() or Directory.Exists()

1. Using Stateful Sessions (Spring)

For modern applications running in the Cloud, it is not recommended to be stateful, especially for sessions as they’re not scalable, and are generally harder to replicate and fix bugs (server-side). Ideally, stateful sessions should be replaced by stateless and client-side mechanisms such as cookies, client cache (e.g. Redis, memcache…) or in an external cloud-based storage. This is an important architectural constraint of microservices-style applications, as it enables resiliency, elasticity, and allows any available service instance to execute any task.

**Pattern:**

**Java:** Look in “WEB-INF/applicationContext.xml” for

<beans>

<bean id=”accountService” class=”com.foo.DefaultAccountService” scope=”session”/>

…

</beans>

**Links:**

Detailed explanation for the topics covered by cast highlight are available at <https://www.casthighlight.com/cloudready-pattern-definitions/>

Detailed report with respect to blockers can be found at <https://www.casthighlight.com/category/product/indicators-methodology/cloudready/blocker/>

**Rules to be added to VSM:**

1. Persistent Files - Using stateful session (Servlet):

**Old Rules:**

Looks for,

(HttpSession http)

new HttpSession(

.setAttribute(

**New Rule:**

Should add these:

import javax.servlet.http.HttpSession;

and

getSession().setAttribute( or getSession().putValue(

1. Security & User Authentication - Using impersonate Identity:

**New Rule:**

Look in source code for import waffle.windows.auth.\*

1. Security & User Authentication - Using of unsecure network protocols (HTTP, FTP):  
   **New Rule:**

Look in source code URL preceded by http:// or ftp://

1. Temporary Files - Access to environment variable:

**New Rule:**

Look in source code for import java.lang.\*; and System.getenv()

1. Perform Directory Manipulation:

**New Rule:**

import org.apache.commons.io.FileUtils; and moveDirectory() or MoveDirectoryToDirectory() or deleteDirectory() or copyDirectory() or copyDirectory() or copyDirectoryToDirectory()…

import java.io.File; and renameTo() or mkdir() or mkdirs() or isDirectory() or delete()

1. Persistent Files - Using stateful session (Spring):

**New Rule:**

Look in “WEB-INF/applicationContext.xml” for  
<beans>  
<bean id=”accountService” class=”com.foo.DefaultAccountService” scope=”session”/>  
…  
</beans>