Adobe ColdFusion (2018 release)

Performance Whitepaper

Executive Summary

One of the focus areas of ColdFusion (2018 release) was to enhance performance to ensure our customers derived the maximum benefit from the ColdFusion runtime.

We selected various ColdFusion functions, tags, applications and frameworks and performed rigorous benchmarking tests to identify the areas of improvement. We used that as an input to optimize the design and implementation of ColdFusion core runtime and language features.

With these changes, our test applications showed an out-of-the-box performance improvement of **30%** and **45%** in throughput, when compared with ColdFusion (2016 release) and ColdFusion 11, respectively.

This whitepaper contains the details of our testing scenarios, graphical representation of the magnitude of improvement observed in various ColdFusion features, the configuration of our test apparatus and the method used to evaluate performance.

CFML Applications and Frameworks

For evaluating the performance of ColdFusion (2018 release), we selected a variety of frameworks and applications, both open-source and proprietary.

The following applications and frameworks were used:

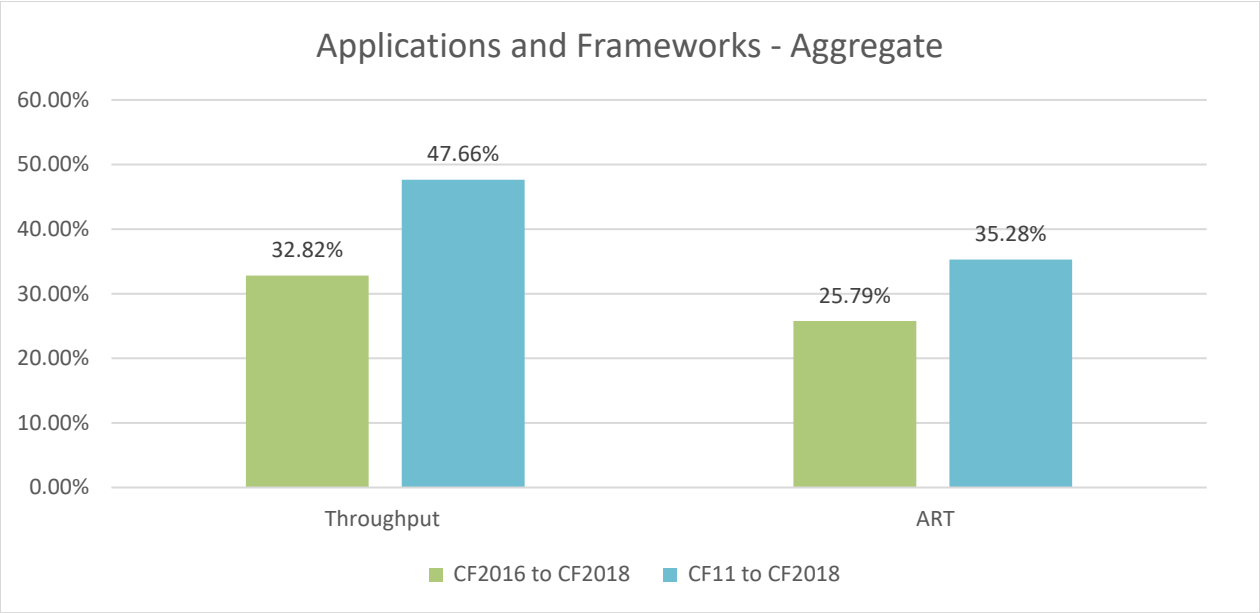
Applications

- Contens - An Enterprise CMS Application
- LearnCFInAWeek – A web-based learning portal for CFML
- BlogCFC – A blogging engine

Frameworks

- Framework One
- ColdBox
- ColdSpring

The following graph shows the improvement in performance when migrating to ColdFusion 2018 from ColdFusion 2016 or ColdFusion 11.



Following is a brief description of the applications, along with a graphical representation of the improvement realized in their performance with ColdFusion (2018 release).

CFML Applications

BlogCFC

BlogCFC is an open-source CFML-based blogging engine that makes extensive use of database queries. We used MS SQL as the test database.

With ColdFusion (2018 release), we see an improvement of **40%** in the throughput and **35%** in the application response time.

LearnCFInAWeek

LearnCFinAWeek is a web-based learning portal for CFML that is widely used by the ColdFusion community.

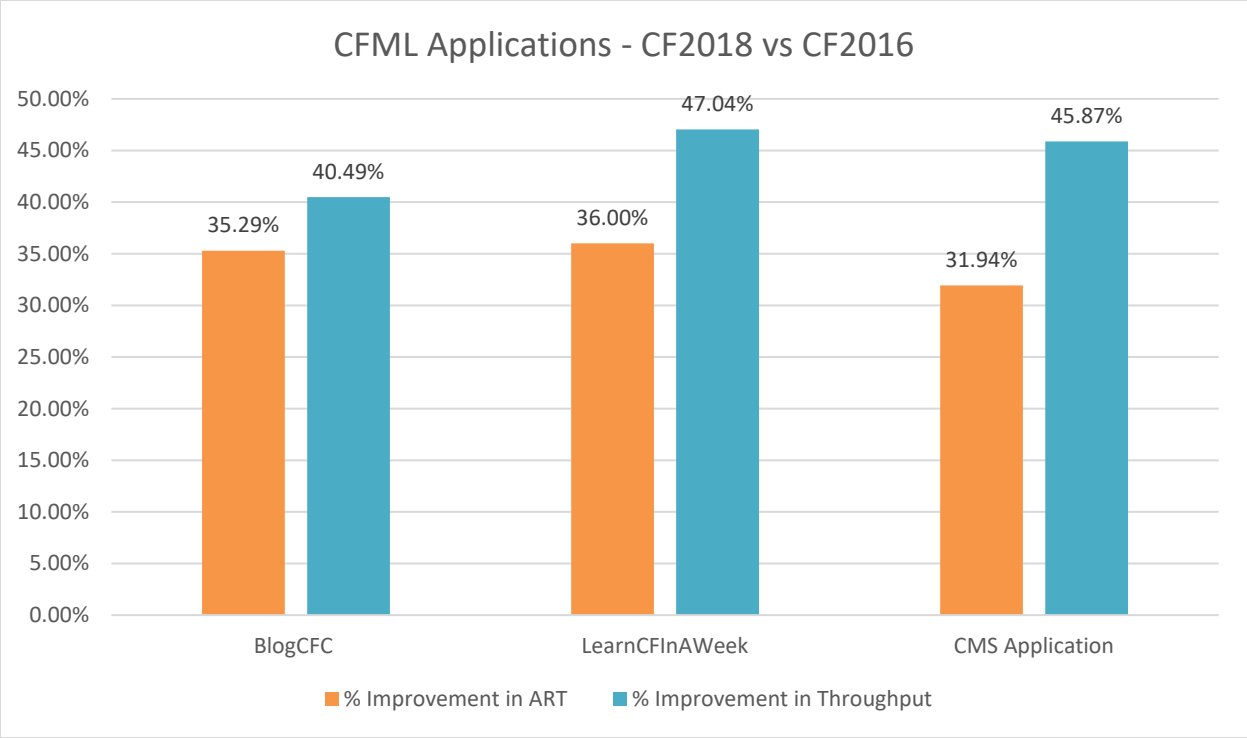
For our benchmarking tests, we replicated the setup in our test environment. The application is backed by content hosted on a MySQL database.

With ColdFusion (2018 release), we see an improvement of **47%** in the throughput and **36%** in the application response time.

Contens CMS

Contens is a Web Content Management platform for websites, intranets and extranets. The application was backed by a MySQL database.

With ColdFusion (2018 release), we see an improvement of **45%** in the application throughput and **32%** in the application response time, when compared with ColdFusion (2016 release).



CFML based Frameworks

Framework One

FW/1 is an MVC-based application framework for CFML applications. It provides a simple, convention-based approach to MVC (Model-View-Controller) applications, as well as REST APIs.

With ColdFusion (2018 release), we see an improvement of **33%** in the throughput when compared with ColdFusion (2016 release).

ColdBox

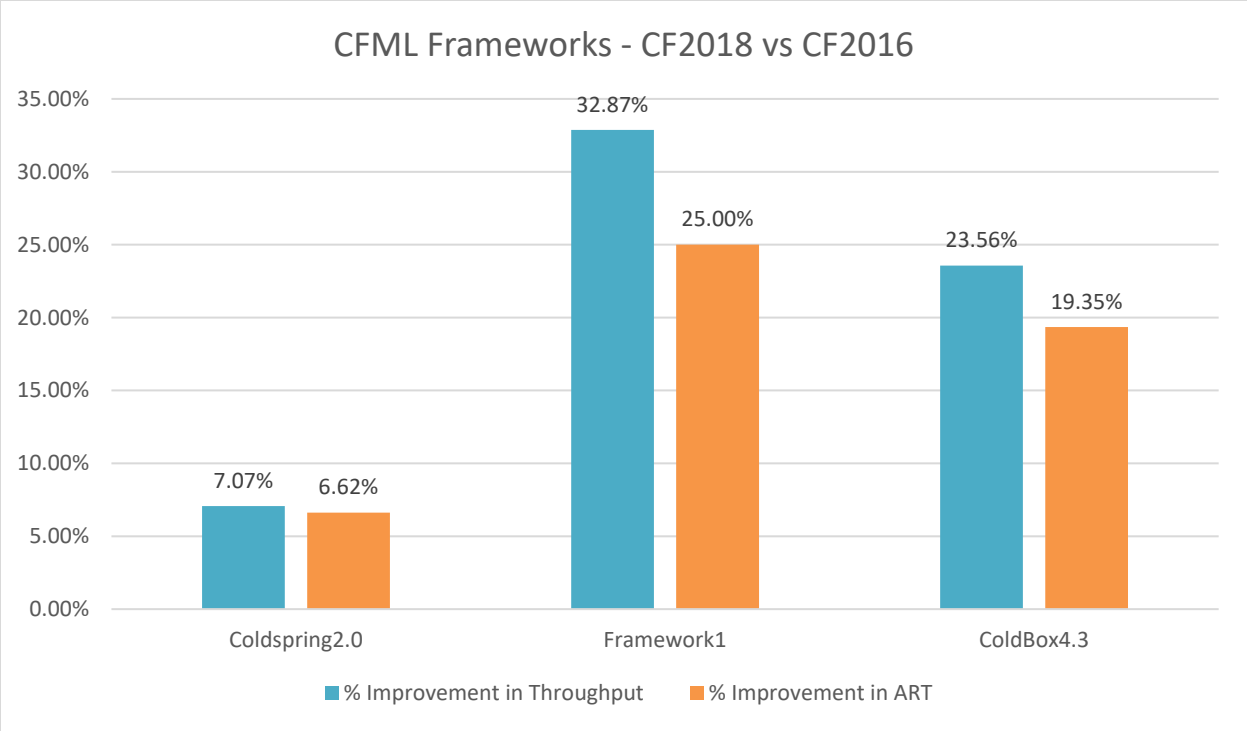
ColdBox is one of the most popular open-source, conventions-based, modular HMVC application framework intended for building enterprise applications with CFML. It uses Model-View-Controller, Dependency injection, and Aspect-oriented programming architectural patterns.

With ColdFusion (2018 release), we see an improvement of **24%** in the throughput when compared with ColdFusion (2016 release).

ColdSpring

ColdSpring is a web application framework for ColdFusion, based on the Java Spring Framework. It provides Dependency injection, inversion of control, and Aspect-oriented programming design pattern for managing dependencies of ColdFusion components (CFCs).

With ColdFusion (2018 release), we see an improvement of **7%** when compared with ColdFusion (2016 release).



Areas of Improvement

Web Server Path Caching

We focused on improving the core ColdFusion runtime performance so that all the applications and frameworks are benefited. To identify the areas where the core engine is slow we profiled the test applications.

One of the areas we optimized is Web Server Path Caching. Whenever a cfm template or CFC is used in an application the template path needs to be translated to the physical path on the disk. Typically, in a production environment the templates path will not change. One can avoid this expensive file I/O call by enabling trusted cache and path cache.

With ColdFusion (2018 release) we have redesigned this caching mechanism for better performance. Earlier we were relying on the container application server for caching these paths. Now we have our own implementation of the mechanism.

Path caching is an enterprise only feature. And many places this path caching has been introduced to improve the core runtime performance.

Web Server Path Caching can only be used in a single website installation. If the application is hosted on multiple sites this cache cannot be used. with ColdFusion (2018 release) we are extending support for web server path caching to multiple website installations as well.

Application CFC.

Irrespective of trusted cache in the request we always search for Application.cfm/cfc. With ColdFusion (2018 release) we are now caching that path when trusted cache is enabled. Now this searching of these paths gets cached for the requested template there by reducing further I/O.

We have also improved the performance of the way the event handlers of application.cfc gets loaded.

SQL Queries

We are now performing a compile time normalization of SQL queries (like whitespace trimming etc.) in the cfquery tag or query functions instead of doing that during query execution.

ColdFusion Mapping / Custom Paths

When an application uses ColdFusion mappings or a custom tag we search through server and application settings involves file I/O which has been optimized.

CFC

CFCs which implement any interfaces are also checked for the validity of their implementation every time they are initialized. We are now caching the results to avoid these checks if the trusted cache is enabled.

Synchronization and Immutability

We are now using granular synchronization instead of hard synchronization. For instance, in structs we are using granular synchronization rather than synchronizing overall data structure.

We are using immutable objects to avoid synchronization and cloning of objects. We are also reusing expensive objects instead of recreating them with every request.

Other changes

- To improve some of the programming constructs we have modified our core runtime to be exception free internally.
- We have moved some of the runtime execution blocks in *cfthrow* to compile time.
- *cfsavecontent* was a custom CF module earlier. It is now a core tag which is more performant.
- We have optimized file IO operations in *cflog*.

ColdFusion Functions and Tags

In ColdFusion, there are more than 600 functions and 140 tags.
We shortlisted 200 functions and tags based on their type. We inferred the following:

- Data structure and decision functions are more likely to be used than for example, image processing or mobile functions
- Publicly available web analytics data
- Data from various ColdFusion learning resources

We analyzed the performance of these functions and tags across different versions of ColdFusion and CFML engines to identify performance improvement opportunities.

Here is an aggregated summary of the improvement observed with functions and tags in ColdFusion (2018 release) when compared with ColdFusion (2016 release).

Function/Tag category	% improvement in execution time
String	48
List	70
Struct	29
Arrays	45
DateTime	60
XML/JSON	71
Decision	51

Following is a graphical representation of our findings. The graphs are categorized according to functional areas.

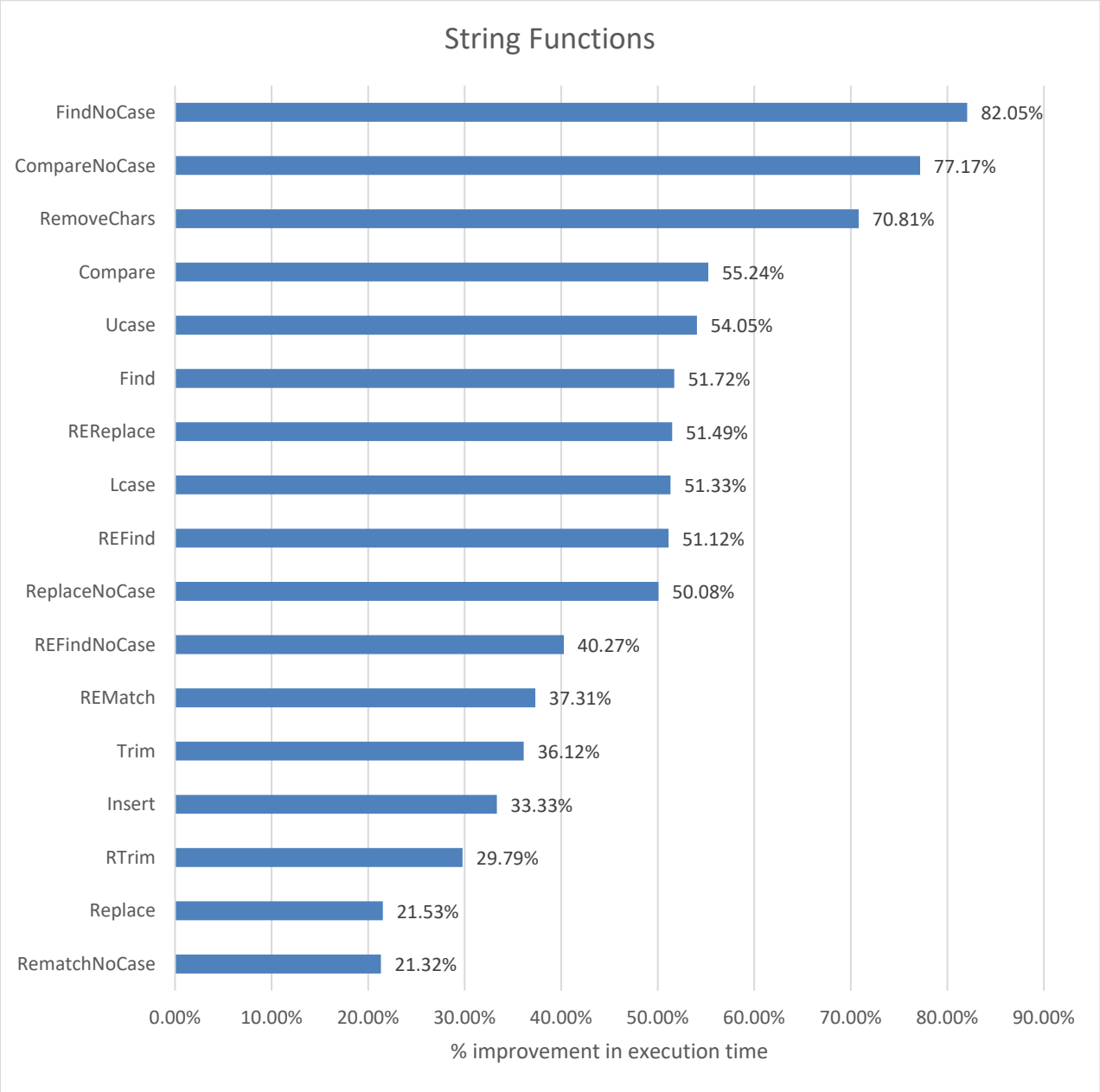
The improvement shown here, is in terms of execution time of the respective function/tag in ColdFusion (2018 release) when compared with ColdFusion (2016 release).

An improvement of 80% implies that the function takes 80% less time to execute in CF 2018 when compared with CF 2016. For example, cfswitch takes 27 ms to execute in CF2018 and 4412 ms to execute in CF2016. This can either be interpreted as a 99.39% improvement [(4412 - 27) / 4412] in execution time in CF 2018 or a 162-fold improvement over CF2016. This document uses the former convention for graphical consistency in charts.

String Functions

String regular expression functions compile the regular expression every time. Now we are caching these regular expressions so that they do not get recompiled every time.

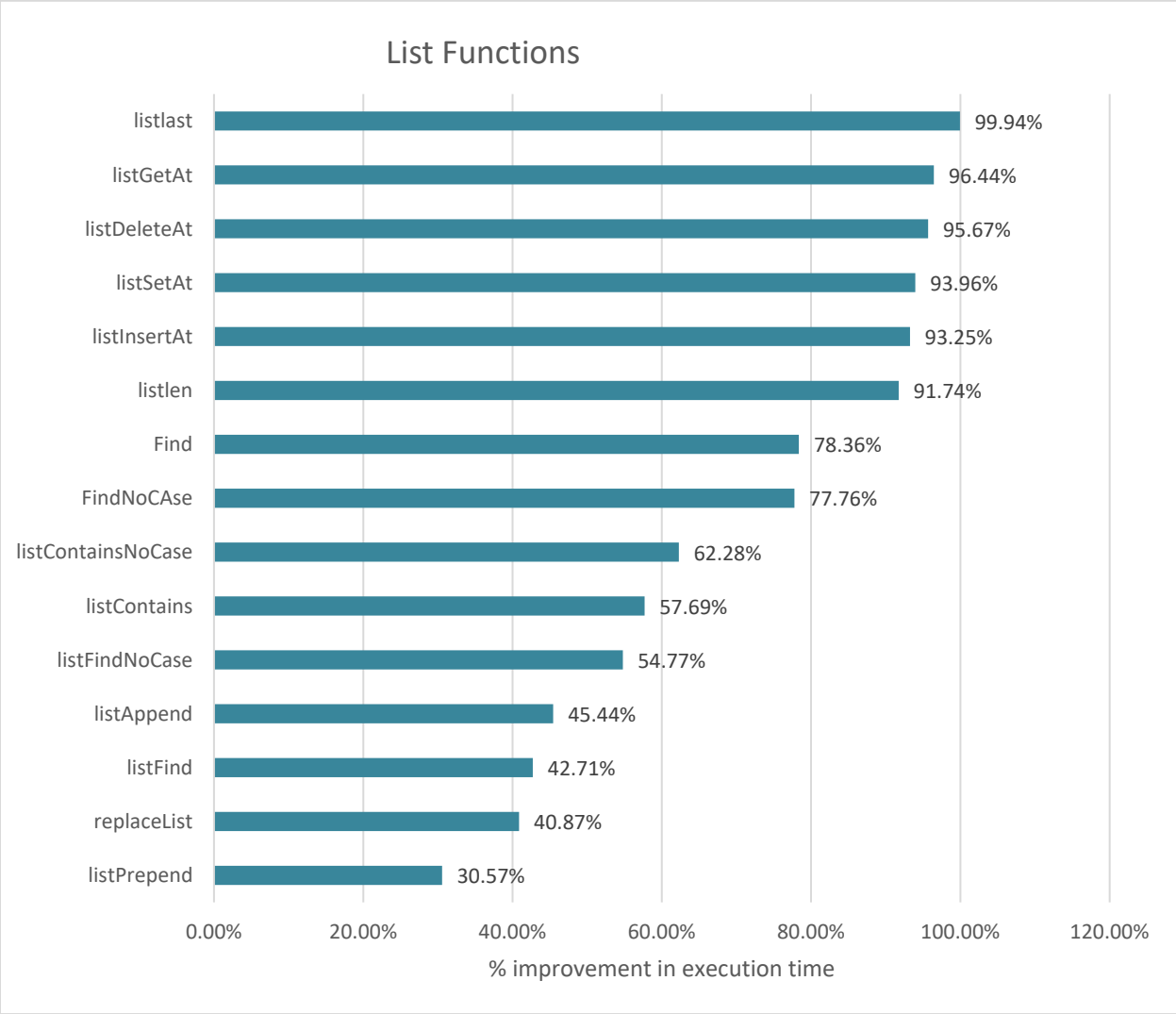
Our tests show that String functions in ColdFusion (2018 release) show **48%** performance improvement. Following are the functions along with the improvement in their respective execution times:



List Functions

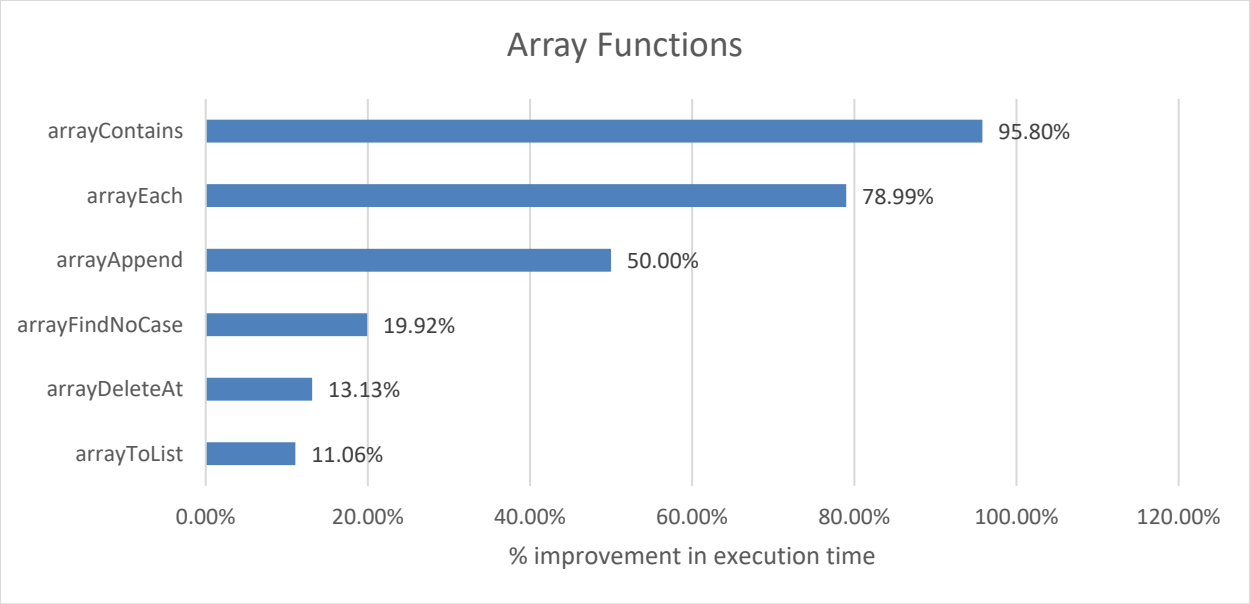
List functions generally use a tokenization algorithm to split the given list before performing the requested operation (like find, insert or delete). We have improved the algorithm for better performance and better memory utilization.

Our tests show that List functions in ColdFusion (2018 release) show **70%** performance improvement. Following are the functions along with the improvement in their respective execution times:



Array Functions

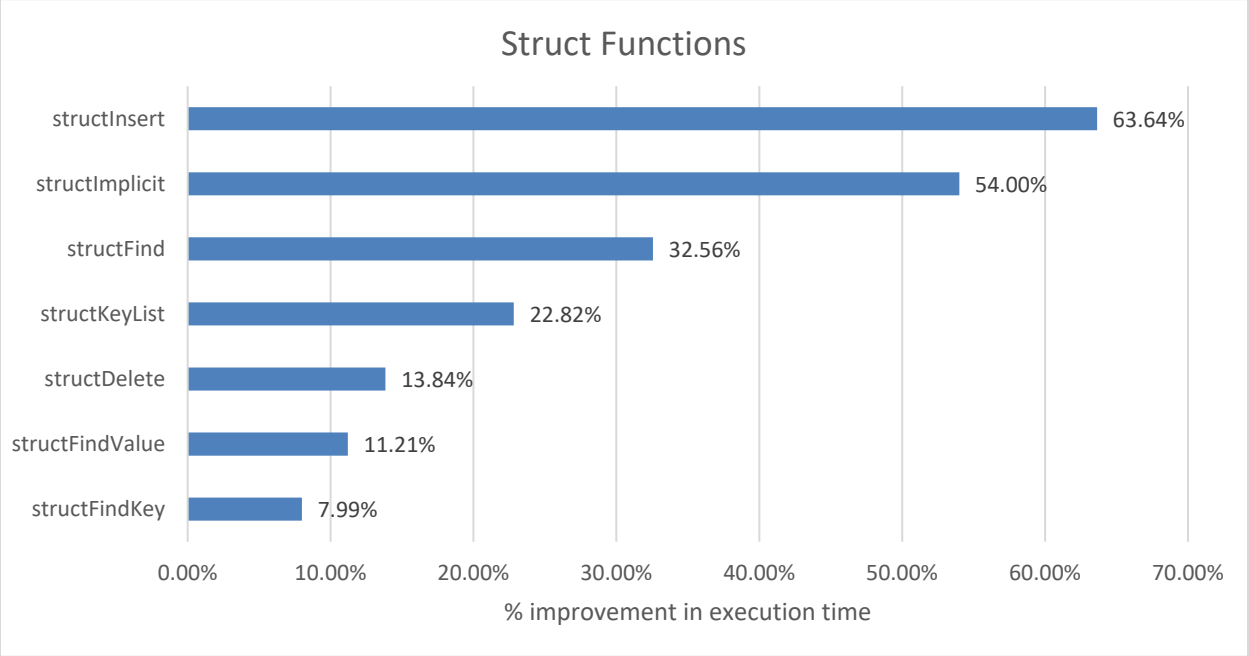
Our tests show that Array functions in ColdFusion (2018 release) show **45%** performance improvement. Following are the functions along with the improvement in their respective execution times:



Struct Functions

We are now using granular synchronization instead of hard synchronization for synchronizing the struct data structure.

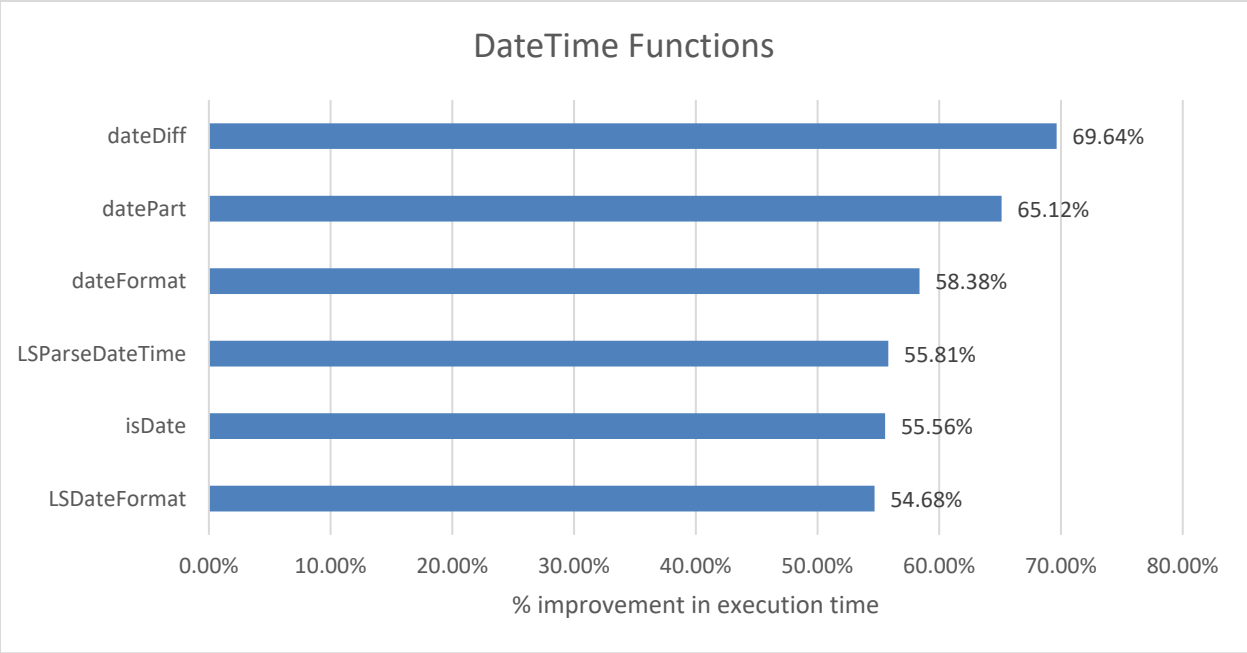
Our tests show that Struct functions in ColdFusion (2018 release) show **30%** performance improvement. Following are the functions along with the improvement in their respective execution times:



DateTime Functions

DateTime conversions used a regex-based approach earlier which was slow. We are now using an improved parsing algorithm.

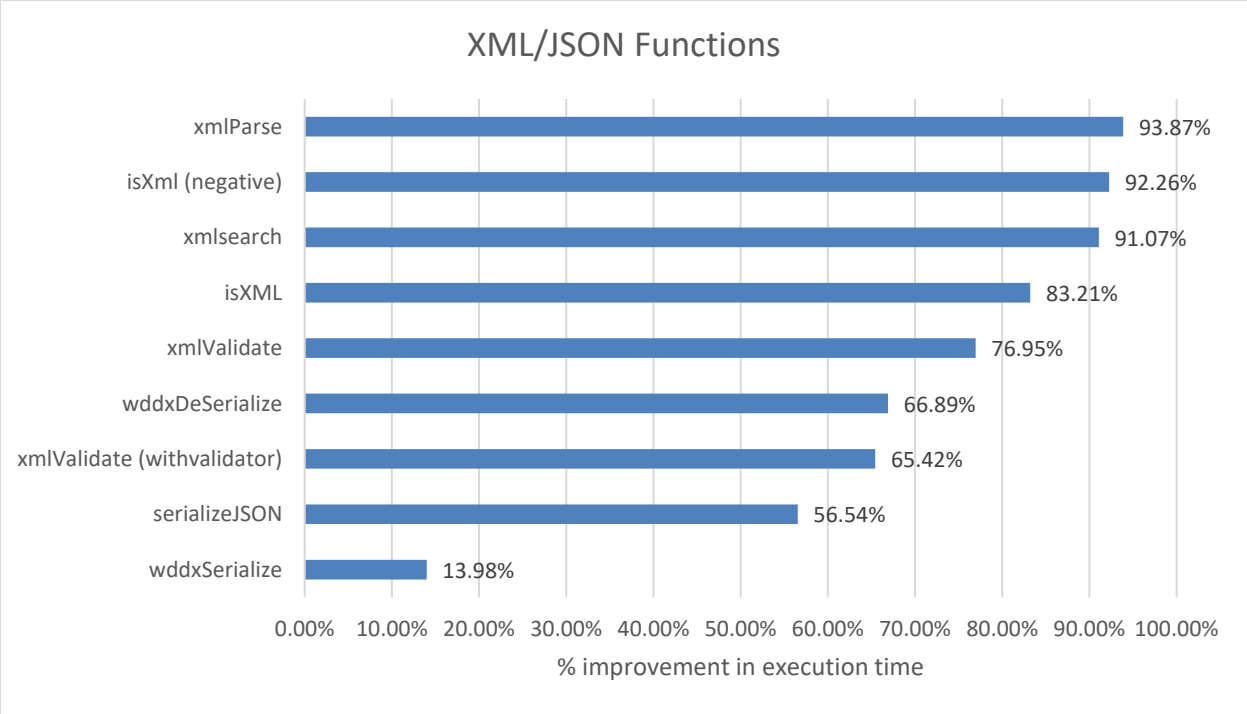
Our tests show that DateTime functions in ColdFusion (2018 release) show **60%** performance improvement. Following are the functions along with the improvement in their respective execution times:



XML Functions

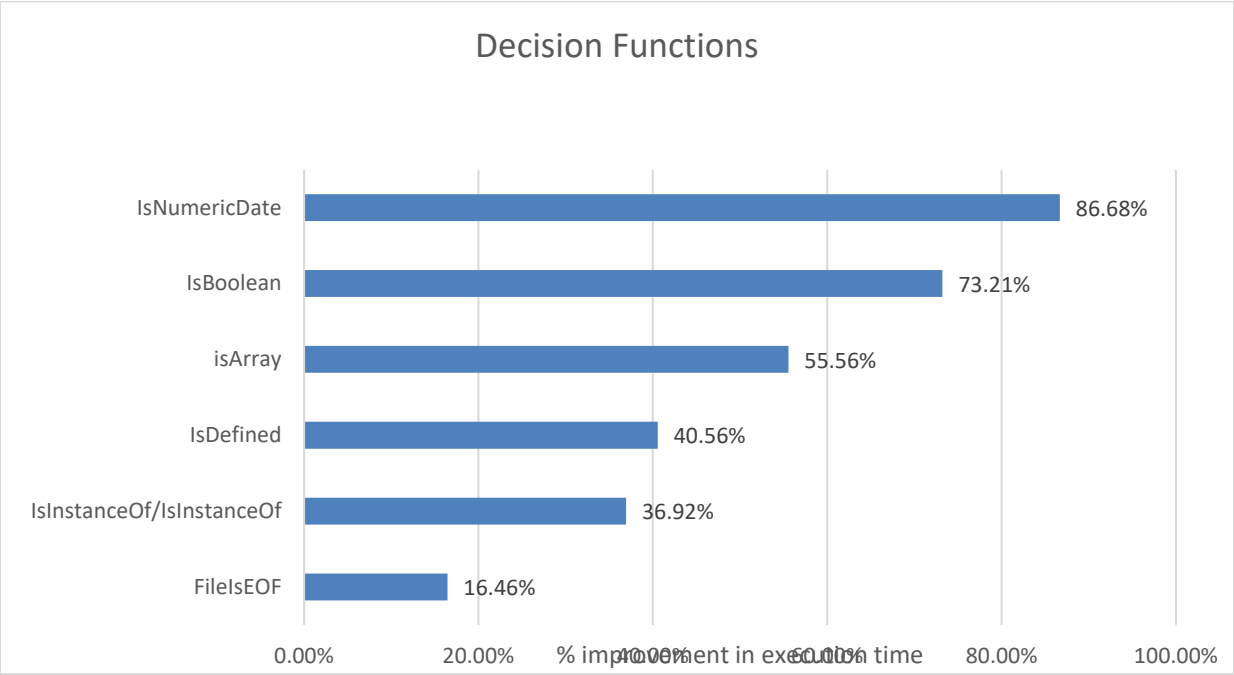
Creation of XML parser instances involves creation, initialization, and setup of many objects that the parser needs and uses for XML document parsing. These initialization and setup operations are expensive. We have made design changes to create parsers once and then reuse the instances while ensuring thread safety.

Our tests show that XML functions in ColdFusion (2018 release) show **71%** performance improvement. Following are the functions along with the improvement in their respective execution times:



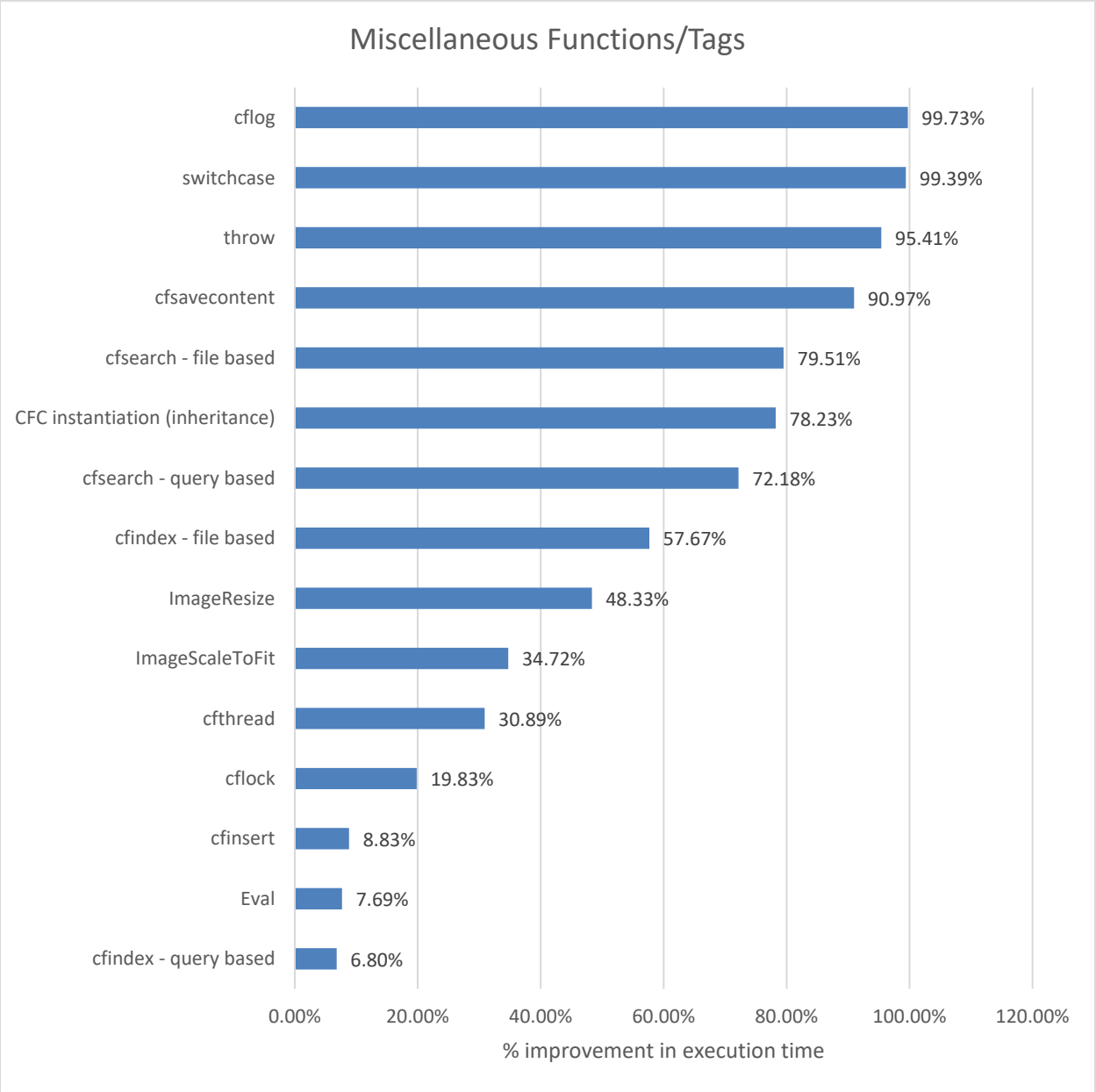
Decision Functions

Our tests show that decision functions in ColdFusion (2018 release) show **51%** performance improvement. Following are the functions along with the improvement in their respective execution times:



Miscellaneous Functions and Tags

Our tests show that decision functions in ColdFusion (2018 release) show **55%** performance improvement. Following are the functions along with the improvement in their respective execution times:



Our approach to performance testing

Performance testing is an intensive exercise that involves tuning different components in a test environment (Operating System, application server, web server, JVM, application, database, etc.) for maximum performance. Performance testing also establishes benchmarks in terms of quantifiable metrics that are reliable and reproducible.

Once the ground work is done, one can perform multiple iterations of load tests to measure performance metrics, identify bottlenecks, fix them, and re-run the cycle to establish improvements.

The following is an overview of the steps involved:

The infrastructure

- Platforms with ample hardware resources (CPU cycles, primary and secondary storage, network bandwidth)
- Configure the following:
    - OS to remove caps that can affect the performance of hosted applications
    - JVM
    - Application server
    - ColdFusion

Hardware configuration

- Dell PowerEdge R720 server
- Windows Server 2016 Standard / RHEL 7.1 x64
- Xeon E5-2637 (4 cores) @ 3.5 GHz – 2 CPU sockets (16 logical cores)
- RAM: 32 GB
- 1 Gbps NIC

The following parameters were modified to allow maximum utilization of network resources by the OS:

- Increase the number of allowed open files
- Increase the TCP port range (1024 - 65535).
- Reduce the TCP time-wait timeout

JVM settings

Configure the JVM parameters in ColdFusion to minimize GC pauses. The heap size should be optimal and set to the same minimum and maximum values to avoid resizing.
JConsole, a GUI-based tool packaged with JDK, can be used to monitor heap usage and GC frequency.

For our testing, we had set the max and min heap size to 2G and the *MaxMetaspaceSize* to 1G

ColdFusion configuration tuning

We had used ColdFusion in production profile that excludes unnecessary servlets and disables debugging by default.

For a simulated load of, say X users, ensure that maximum number of simultaneous template requests is set to at least the same value.

Also ensure that simultaneous Web Service requests, CFC function requests, and number of threads available for *cfthread* are set to an optimal value.

The *maxThreads* attribute for the HTTP connector (used if internal CF port is in use) and AJP connector (in case web server connector is in use) must be set to at least the same value as the number of concurrent user threads that are simulating the load.

In Tomcat, the default value for *maxThreads* is 200.

We enabled the following caching options:

- Trusted cache
- Cache template in request
- Component cache
- Cache web server paths

Building the test plan

It's important to use workflows that are relevant to the application being tested. We used JMeter as the load simulating tool.
For the test applications, the test plans were created with *JMeters test script recorder*. For the frameworks, we used the sample applications that were bundled with the frameworks.

Simulating test loads

The CPU utilization, memory utilization, disk I/O, and network traffic metrics can be monitored with tools like *vmstat* and *top* on Linux, or tools like *Task Manager* and the *Performance Monitor* on Windows.

There are a few things to check when monitoring the system resources. The CPU run queue should not exceed the number of CPU cores. A higher CPU utilization generally translates to better throughput. In terms of percentages of CPU time, for an ideal application, the User time must be higher than the System time.

Before collecting the samples, ensure that all required classes are compiled by hitting the target URL at least once. We ran the load tests for short warm-up intervals to allow for the JIT optimizations to kick in and the throughput to stabilize. We collected multiple samples, with a typical sample interval of 120 secs.

The performance metrics that were monitored are:

- Throughput (the number of transactions per unit time)
- ART (the average amount of time it takes to process one transaction)

The error percentage must always be zero..

The *keep-alive* must be set in the HTTP sampler so that connections between round trips are persisted. A new connection for every single request may strain the host system unnecessarily.

Use the *Cookie Manager* configuration element so that cookies are stored and re-used in subsequent requests.

For baselining the environment, we used a simple HelloWorld CFML application that produces consistent metrics across multiple runs.

When executing the load tests, ensure that you:

- Use JMeter in console mode
- Use minimal number of listeners
- Use assertions to ensure that there are no unexpected responses
- Monitor the error to ensure that it is zero or an acceptable value

Approach to optimizing performance

We used an iterative approach to improvement by simulating load on test application, using CPU profiling tools such as *jVisualVM* (a standard JDK tool) to identify performance bottlenecks.

We eliminated those bottlenecks by redesigning the workflow wherever possible. We then re-run the load tests to confirm the fix and identify other bottlenecks.

Appendix

Hardware/ColdFusion/Test Configuration

Testing configuration and tools	JVM settings	Server settings
• Dell PowerEdge R720 server • Windows Server 2012 R2 • Xeon E5-2637 (4 cores) @ 3.5 GHz – 2 CPU sockets • RAM: 32 GB • Client: Apache jMeter-4.0 100 concurrent users • Database: Microsoft SQL Server 2012 • MySQL 5.7	• Java: JRE bundled with ColdFusion • Min JVM heap size: 2048m • Max JVM heap size: 2048m • MaxMetaspace 1024m • Parallel GC	• Max no. of simultaneous templates/CFCs/cfthread: 100 • Whitespace Management: enabled • Trusted cache: enabled • Component cache: enabled • Web server path cache: enabled • RDS disabled • Line debugging disabled

Application/ Framework	Version	Sample applications used
ColdFusion 2016	2016.0.06.308055	N/A
ColdFusion 11	11.0.14.307976	N/A
BlogCFC	1.7	N/A
Contens	5	N/A
Framework One	4.3	Helloworld, Hellolinked, Hellolayout, Hellocontroller, Helloservice, Modular, Remote, UserManager, UserManagerAJAX
ColdSpring	2.0	CS in 5mins, factoryBeans, AOPEg
ColdBox	4.3	ColdBoxSES, FeedGenerator, feedReader, i8NSamplem and javaLoader