BOXLANG®

Overview

Ortus Solutions

Ortus Solutions

BOXLANG®

DYNAMIC : MODULAR : PRODUCTIVE

BoxLang is a **modular dynamic** language for the JVM, aiming to make your development more productive, expressive, functional, and available **everywhere**.

BOXLANG®

Ortus Solutions

Goals & Vision

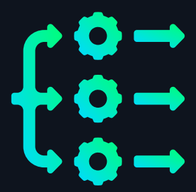
Goals & Vision

Ortus Solutions

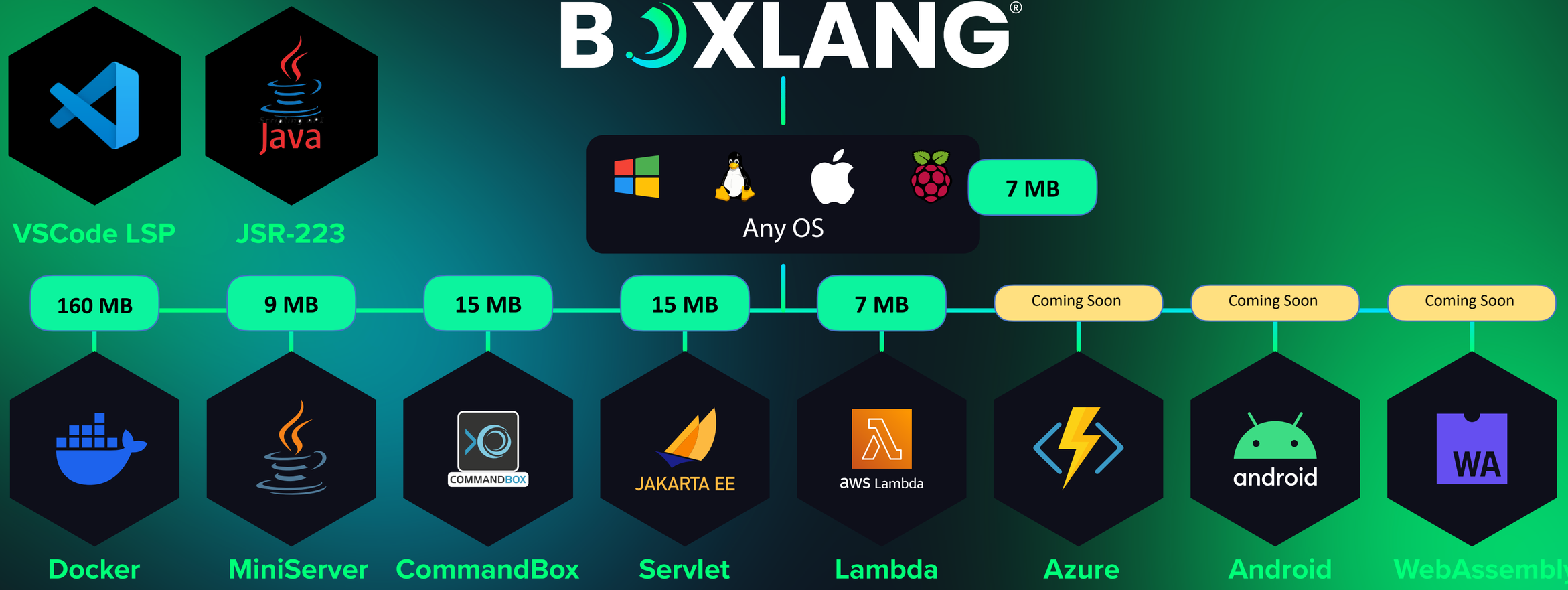
- Be dynamic, modular, lightweight, and fast
- Be 100% interoperable with Java
- Be modern, functional, and fluent
  (Think mixing CFML, Node, Kotlin, Java, and Clojure)
- Modularity at its core
- Take advantage of the modern JVM
- TDD: Fully tested source code
- Be able to support multiple runtimes
- Have multiple transpilers
  CFML -> BoxLang,
  Groovy -> BoxLang
  X -> BoxLang
- IDE and Tools
- All of our libraries needed to run
- Compete in today’s language environments

Ortus Solutions

Multi-Runtime Architecture



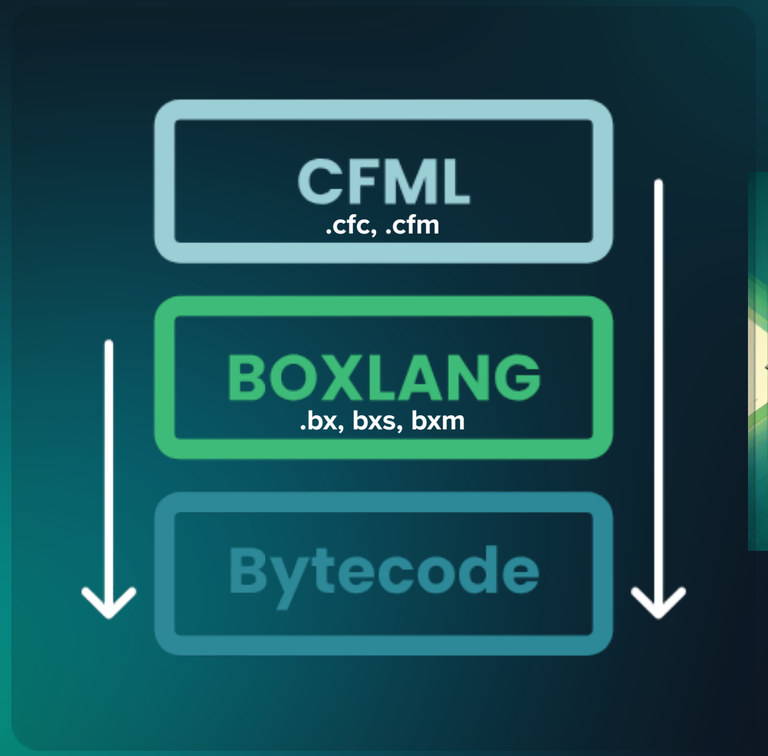
Multi-Runtime Architecture



Multi-Parsers

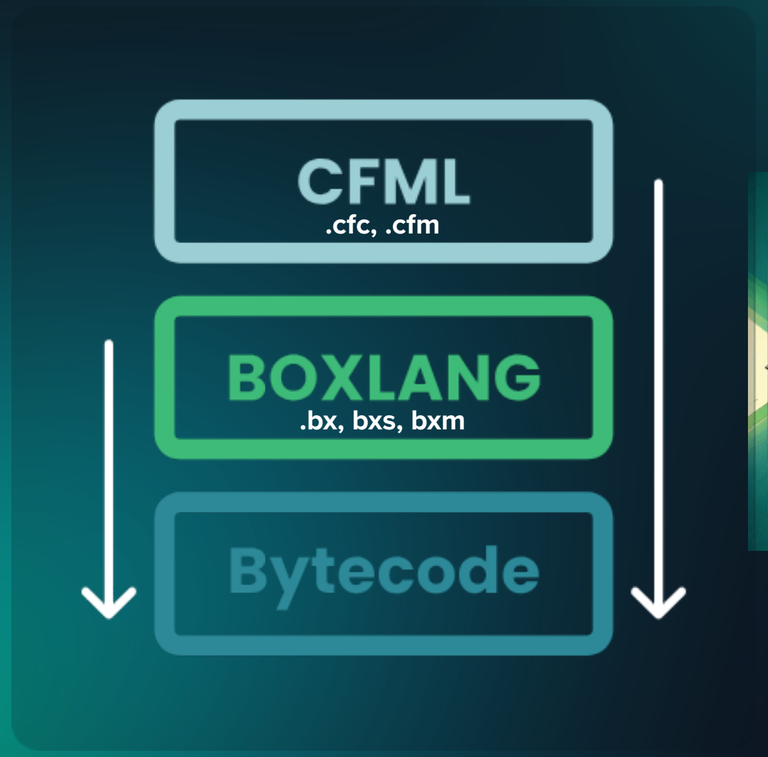
Multi-Parsers : BoxLang + CFML + ???

- **Our way to split with the old and bring in the new**
- Transpile CFML into BoxLang
- BoxLang is a NEW clean slate
- Our way to provide static analysis tooling
- Tons of features now in our IDE

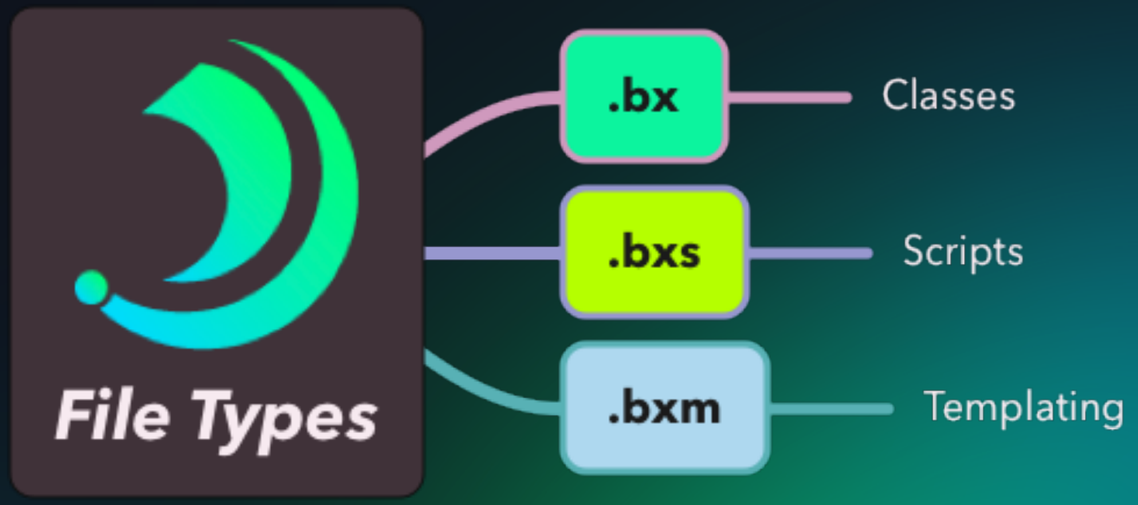


CFML Drop-In Replacement

- For Drop-In Replacement of CFML
  - bx-compat-cfml Module
- Tools
  - CFML Feature Audit
    - `boxlang featureAudit <options>`
    - `boxlang cftranspile <options>`
  - CFML to BoxLang Transpiler



New File Types

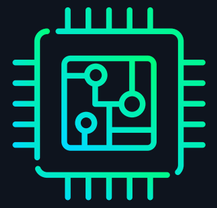


CFML Types

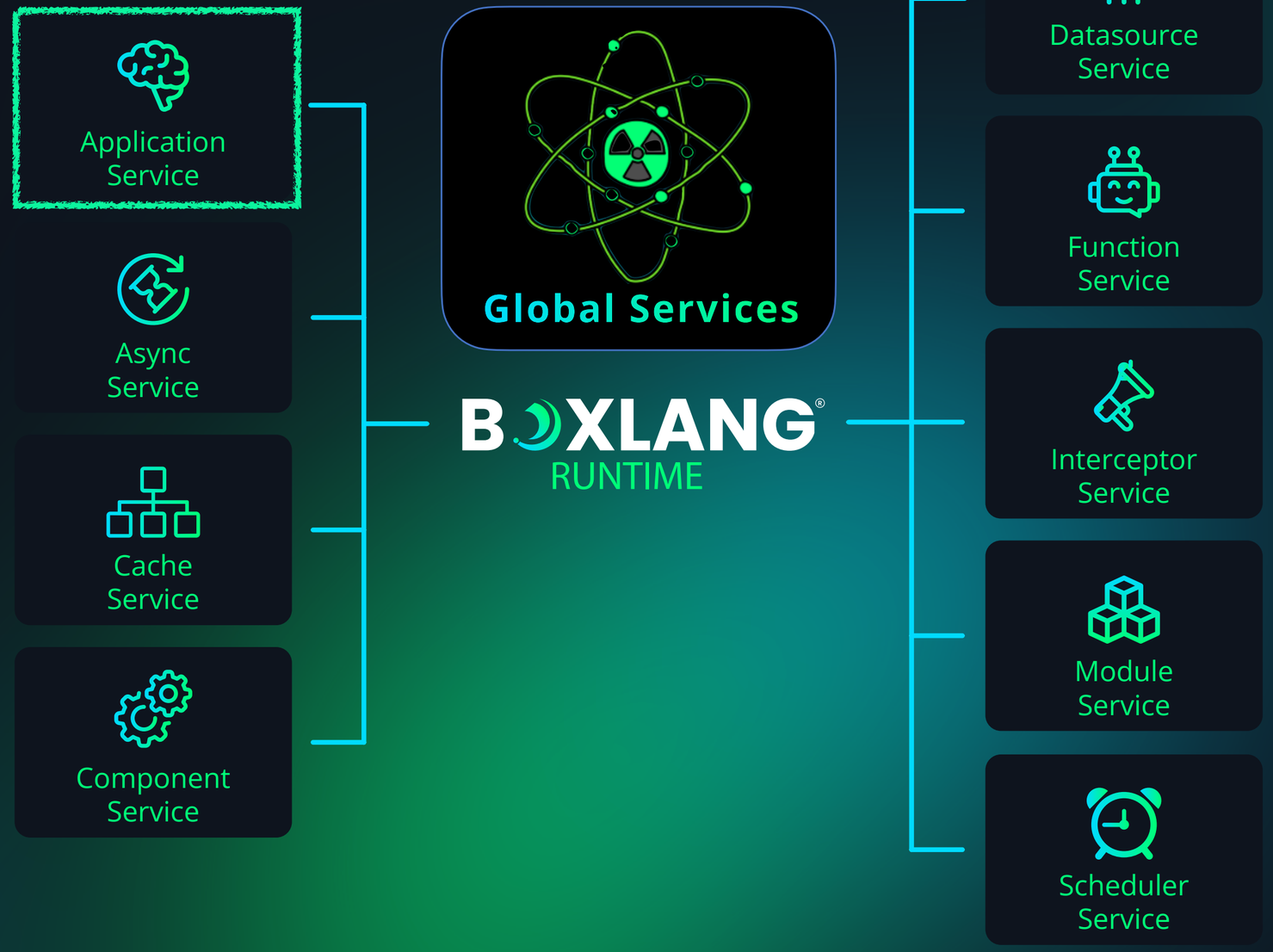
- .cfc
- .cfs — Lucee Only
- .cfm

Ortus Solutions

BoxLang Framework



BoxLang Framework

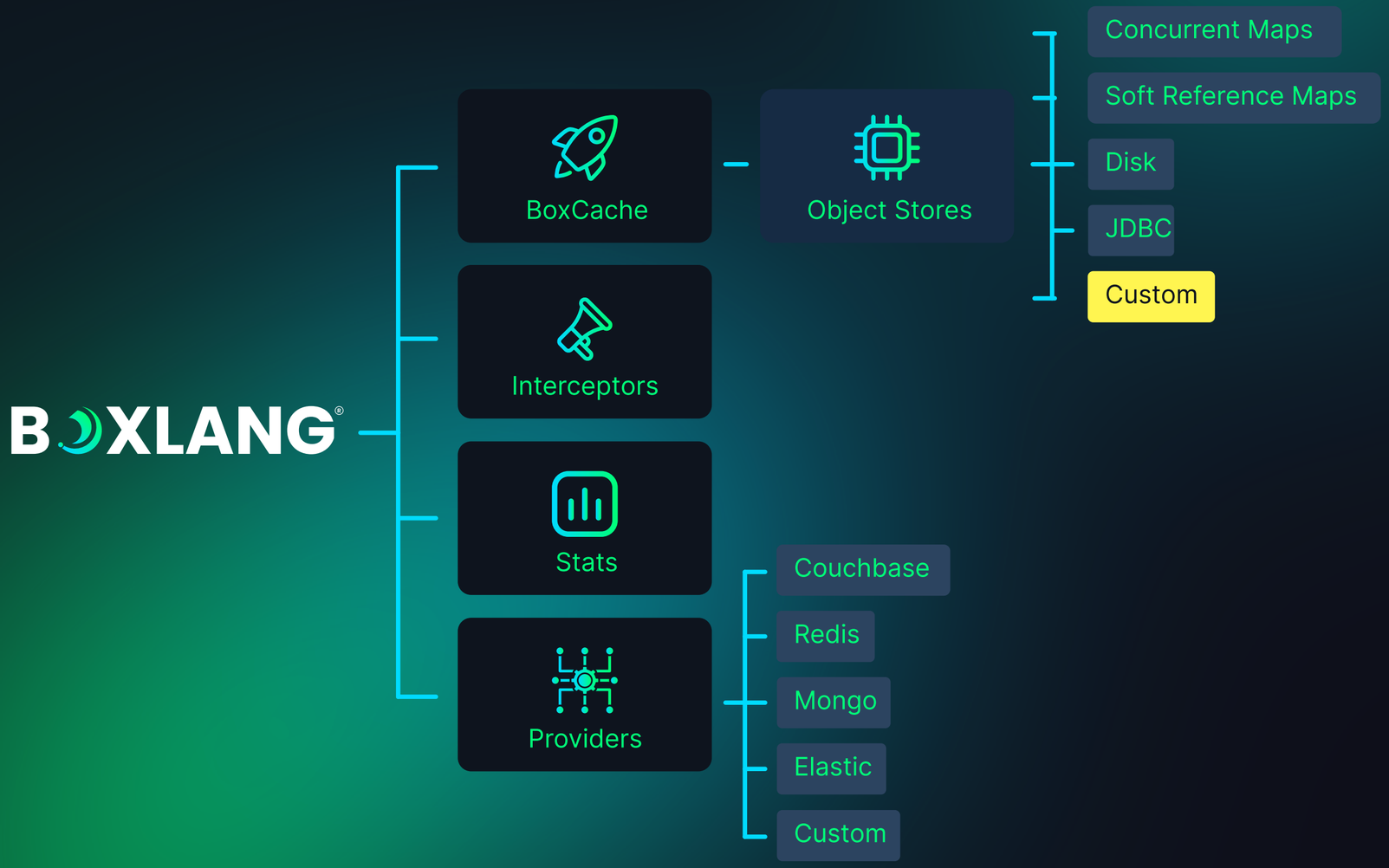


Ortus Solutions

Enterprise Caching Engine & Aggregator

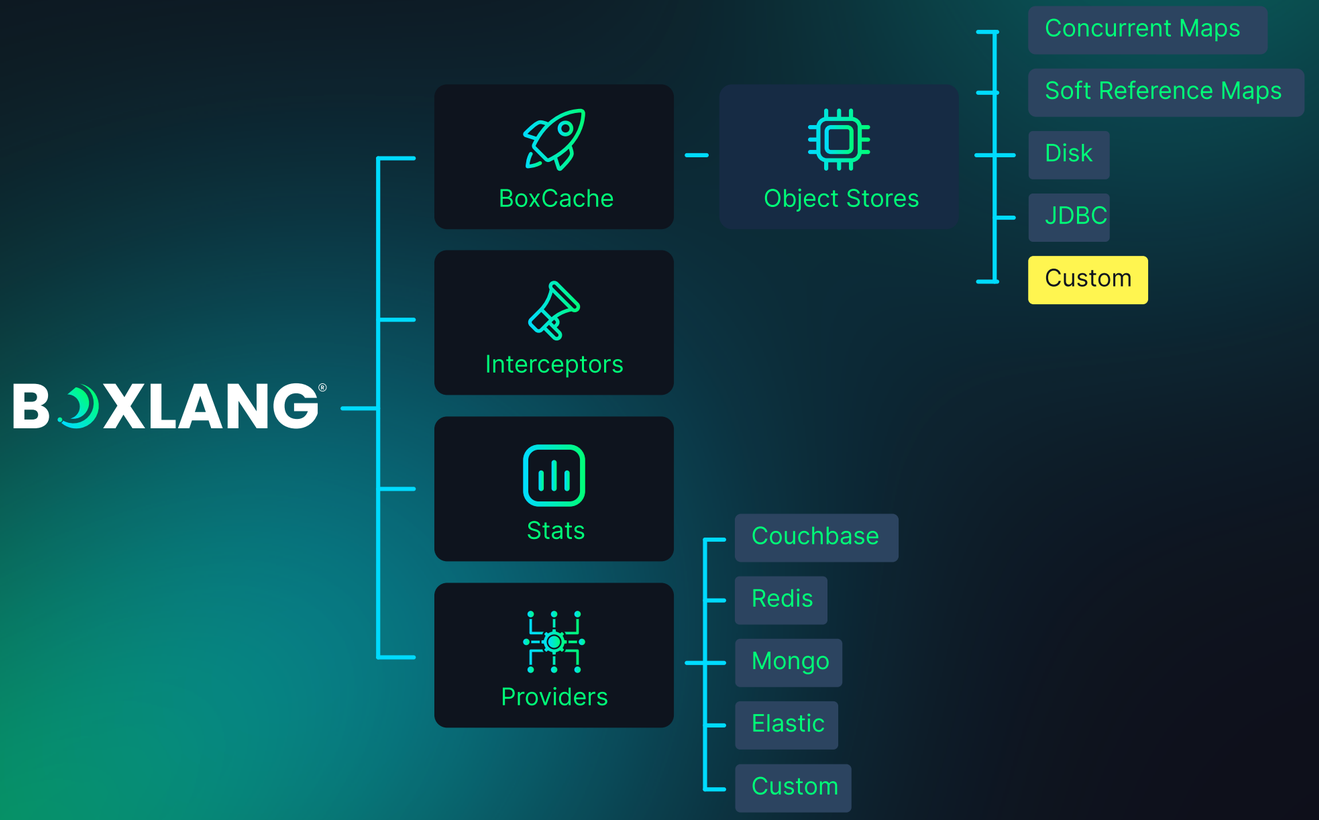
Enterprise Caching Engine & Aggregator

- Inspired by CacheBox
- Enterprise Caching Engine
- Extensible
  - Custom providers
  - Custom object stores
  - Listeners
- Stats
- Powers all internal caching
- Managed by the **CacheService**



Enterprise Caching Engine & Aggregator

- Pre-Defined Caches
  - **Default**: Queries, templates, anything you want.
  - **bxSessions**: Application sessions are stored
  - **bxRegex**: Compiled dynamic regular expressions



Default Properties

- Defaults for all BoxCacheProviders
- Find them in ~/.boxlang/config/boxlang.json

```
// How many to evict at a time once a policy is triggered
"evictCount": 1,
// The eviction policy to use: Least Recently Used
// Other policies are: LRU, LFU, FIFO, LIFO, RANDOM
"evictionPolicy": "LRU",
// The free memory percentage threshold to trigger eviction
// 0 = disabled, 1-100 = percentage of available free memory in heap
// If the threadhold is reached, the eviction policy is triggered
"freeMemoryPercentageThreshold": 0,
// The maximum number of objects to store in the cache
"maxObjects": 1000,
// The maximum in seconds to keep an object in the cache since it's last access
// So if an object is not accessed in this time or greater, it will be removed from the cache
"defaultLastAccessTimeout": 1800,
// The maximum time in seconds to keep an object in the cache regardless if it's used or not
// A default timeout of 0 = never expire, careful with this setting
"defaultTimeout": 3600,
// The object store to use to store the objects.
// The default is a ConcurrentStore which is a memory sensitive store
"objectStore": "ConcurrentStore",
// The frequency in seconds to check for expired objects and expire them using the policy
// This creates a BoxLang task that runs every X seconds to check for expired objects
"reapFrequency": 120,
// If enabled, the last access timeout will be reset on every access
// This means that the last access timeout will be reset to the defaultLastAccessTimeout on every access
// Usually for session caches or to simulate a session
"resetTimeoutOnAccess": false,
// If enabled, the last access timeout will be used to evict objects from the cache
"useLastAccessTimeouts": true
```

Cache Events

- Events are emitted by the CacheService, CacheProvider and ObjectStores
- Cache Service
  - `afterCacheServiceStartup`
  - `beforeCacheServiceShutdown`
  - `afterCacheServiceShutdown`
- Object Stores
  - `afterCacheElementInsert`
  - `beforeCacheElementRemoved`
  - `afterCacheElementRemoved`
  - `afterCacheElementUpdated`
- CacheProviders
  - `afterCacheClearAll`
  - `afterCacheRegistration`
  - `afterCacheRemoval`
  - `beforeCacheRemoval`
  - `beforeCacheReplacement`
  - `beforeCacheShutdown`
  - `afterCacheShutdown`

Default Cache BIFs

- `cache( name ) : ICacheProvider`
- `cacheFilter( filter, [useRegex=false]) : ICacheKeyFilter`
- `cacheNames() : array`
- `cacheProviders() : array`
- `cacheService() : CacheService`

https://boxlang.ortusbooks.com/boxlang-language/reference/built-in-functions/cache

Common Cache Methods

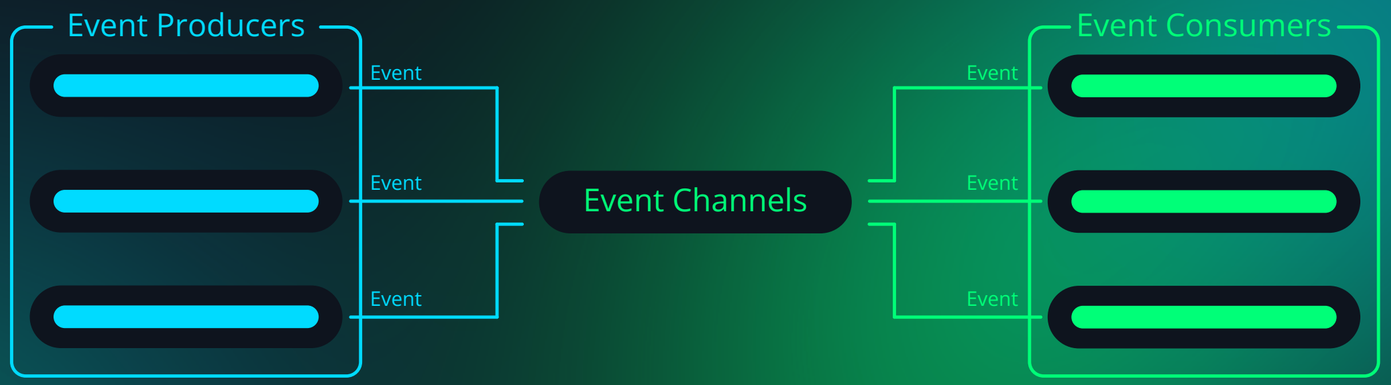
- clear(key):boolean
- clearAll([filter:ICacheKeyFilter]):void
- clearStats()
- get(key):Attempt
- get(ICacheKeyFilter):struct
- getConfig():CacheConfig
- getAsync(key):BoxFuture
- getCachedObjectMetadata(key):struct
- getKeys([filter:ICacheKeyFilter]):array
- getKeysStream([filter:ICacheKeyFilter]):stream
- getName():Key
- getOrSet(key, lambdaOrClosure):any
- getSize([ICacheKeyFilter])
- isEnabled()
- isReportingEnabled()
- getStats():ICacheStats
- getStoreMetadataReport([limit=0=])
- lookup(key):boolean
- lookup(filter:ICacheKeyFilter):boolean
- set(key, value, timeout, lastAccessTimeout, metadata)
- set(entries:struct, timeout, lastAccessTimeout)

https://apidocs.ortussolutions.com/boxlang/1.0.0/ortus/boxlang/runtime/cache/providers/BoxCacheProvider.html

Event-Driven Language

Event-Driven Language

- Interceptors for the language, application, and request
- The best way to scale the language
- Listen to the entire or specific language life-cycles
- Modules can listen/collaborate events
- Just like ColdBox Interceptors




```
// announce the scope creation
BoxRuntime.getInstance().announce(
    BoxEvent.ON_SERVER_SCOPE_CREATION,
    Struct.of(
        ...values:"scope", this,
        "name", ServerScope.name
    )
);
```

src > main > bx > interceptors > LuceeServerScope.bx

```
25  class{
31      /**
38      function onServerScopeCreation( data ){
39          // Add coldfusion struct
40          data.scope.put(
41              Key.coldfusion,
42              new ImmutableStruct( {
43                  "productlevel" : "os",
44                  "productname" : "Lucee",
45                  "productversion" : "2016,0,03,300357"
46              } )
47          );
48
49          // Add lucee struct
50          data.scope.put(
```

Unique Event Pools

Global Runtime

Application.bx
Request Listener

CacheProviders

3 Unique Pools you can register interceptors with

BIFS

Announce

- `boxAnnounce()`
- `boxAnnounceAsync()`

Registration

- `boxRegisterInterceptionPoints()`
- `boxRegisterInterceptor()`
- `boxRegisterRequestInterceptor()`
- `boxUnregisterInterceptor()`
- `boxUnregisterRequestInterceptor()`

Access to Pools

- `getBoxRuntime().getInterceptorService()`
- `getCache( name ).getInterceptorPool()`

Interceptors

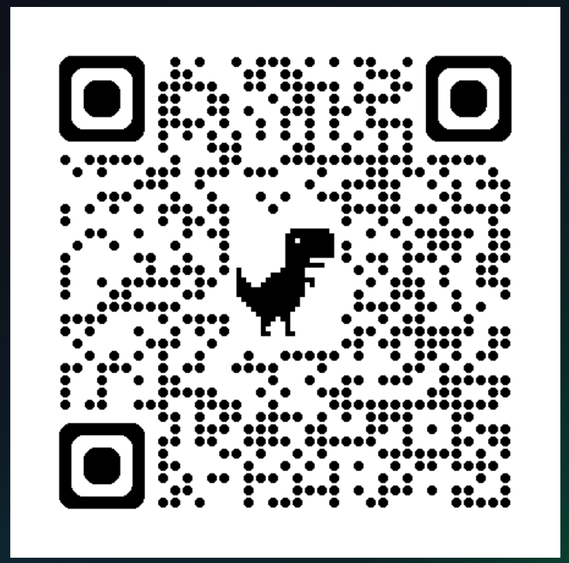
- Can be:
  - Java Classes
    - 1+ events
  - BoxLang Classes
    - 1+ events
  - Closures / Lambdas
    - 1 event
- Name of the event is the name of the method
- Struct of event data
- Use our registration BIFs or directly on a pool

```boxlang
/**
 * BoxLang Class Interceptor
 */
class{

    function onHTTPResponse( struct data ){
    }

}
```

Core Interception Events



https://boxlang.ortusbooks.com/boxlang-framework/interceptors

- Interceptors
  - Core Interception Points
    - Application Events
    - Cache Service Events
    - Cache Provider Events
    - Cache Object Store Events
    - Datasource Service Events
    - Dump Events
    - Dynamic Object Events
    - Function Invocations
    - HTTP Events
    - Life-cycle Events
    - Logging Events
    - Module Events
    - Module Service Events
    - Object Marshalling Events
    - Query Invocations
    - Runtime Events
    - Request Context Events
    - Scheduler Events
    - Scheduler Service Events
    - Template Invocations
    - Transaction Events

Ortus Solutions

Scheduling & Task Framework

Scheduling & Task Framework

- Write schedulers in BoxLang, Java, or both!
- Use in CLI, Web, Lambda, Etc
- Global Schedulers
  - Configuration
- Application.bx Schedulers
  - `this.schedulers = [ “path.to.Scheduler” ]`
- At Runtime Schedulers
  - `BIFs`

```
class {

    // Properties
    property name="scheduler";
    property name="runtime";
    property name="logger";
    property name="asyncService";
    property name="cacheService";
    property name="interceptorService";

    /**
     * The configure method is called by the BoxLang runtime
     * to allow the scheduler to configure itself.
     *
     * This is where you define your tasks and setup global configuration.
     */
    function configure(){
        // Setup Scheduler Properties
        scheduler.setSchedulerName( "My-Scheduler" )
        scheduler.setTimezone( "UTC" )

        // Define a lambda task
        scheduler.task( "My test Task" )
            .call( () -> {
                println( "I am a lambda task: #now()#" );
            } )
            .every( 2, "second" );
    }
```

Highly Configurable

```
"scheduler": {
    // The default scheduler for all scheduled tasks
    // Each scheduler can have a different executor if needed
    "executor": "scheduled-tasks",
    // The cache to leverage for server fixation or distribution
    "cacheName": "default",
    // An array of BoxLang Schedulers to register upon startup
    // Must be an absolute path to the scheduler file
    // You can use the ${user-dir} or ${boxlang-home} variables or any other environment variable
    // Example: "schedulers": [ "/path/to/Scheduler.bx" ]
    "schedulers": [],
    // You can also define tasks manually here
    // Every task is an object defined by a unique name
    // The task object is a struct with the following properties:
    // - `crontime:string` - The cron time to run the task (optional), defaults to empty string
    // - `eventhandler:path` - The absolute path to the task event handler(optional), defaults to empty string
    // - `exclude:any` - Comma-separated list of dates or date range (d1 to d2) on which to not execute the scheduled task
    // - `file:name` - Name of the log file to store output of the task (optional), defaults to `scheduler`
    // - `group:string` - The group name of the task (optional), defaults to empty string
    "tasks": {}
  },
```

Dynamic Interaction

- BIFS:
  - SchedulerStart(path, [force=false])
  - SchedulerShutdown(name, [force=false], [timeout=0])
  - SchedulerRestart(name, [force=false], [timeout=0])
  - SchedulerStats([name])
  - SchedulerList()
  - SchedulerGet(name)
  - SchedulerGetAll()
  - getBoxRuntime().
    - getAsyncService()
    - getSchedulerService()

```
class {

    // Properties
    property name="scheduler";
    property name="runtime";
    property name="logger";
    property name="asyncService";
    property name="cacheService";
    property name="interceptorService";

    /**
     * The configure method is called by the BoxLang runtime
     * to allow the scheduler to configure itself.
     *
     * This is where you define your tasks and setup global configuration.
     */
    function configure(){
        // Setup Scheduler Properties
        scheduler.setSchedulerName( "My-Scheduler" )
        scheduler.setTimezone( "UTC" )

        // Define a lambda task
        scheduler.task( "My test Task" )
            .call( () -> {
                println( "I am a lambda task: #now()#" );
            } )
            .every( 2, "second" );
    }
```

CFML COMPAT

With Brad Wood

CFML Compatibility

Ortus Solutions

BOXLANG®

- Dual Parser: 99.999999%
- Core BIFS: 100%
- Core Tags: 100%
- Anything else, we need you to test!

BIFS

Tags

CFML

Modern Dynamic Language

- Dynamically typed just like CFML, but we go further...
- JDK21+ Minimum
- Fully JSR-223 Compliant
    - Clojure + BoxLang in development by Sean Corfield
- No reflection, we use **InvokeDynamic for everything**
    - **DynamicObject: Any Object can be Dynamic!**
- All OO Constructs
    - Interfaces, superinterfaces and default method implementations
    - Abstract classes and methods
    - Static scope and methods on classes and interfaces
- Use all-new JDK features and types
- Collection of Dynamic Casters and Helpers

Java Interop

- Interact with Java naturally
- It's just part of the language; no more separation
- Type inference, auto-casting, type promotions and coercion
  - Long -> Doubles, Doubles ->Longs, etc
  - BoxLang Function -> Java Lambdas
- You can import, extend, implement, annotate from Java

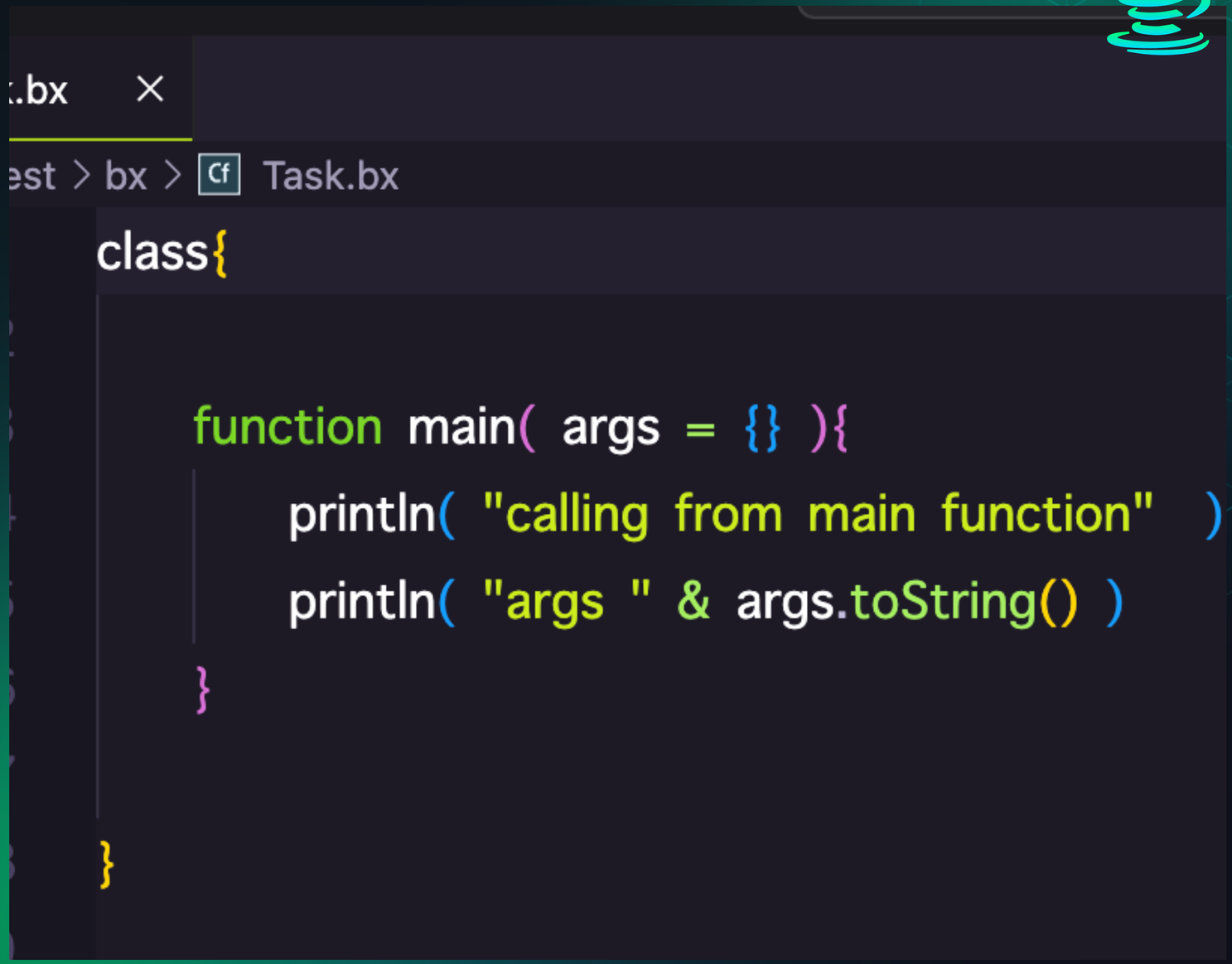
generatePrimes.cfs

rc > test > bx > generatePrimes.cfs

```
1  import java.lang.System
2  import java.util.Date as MyDate
3  import ortus.boxlang.runtime.types.Array
4
5  start = new MyDate().getTime()
6  num = 1000
7  myArray = []
8  arr = Array.copyOf( myArray );
9  System.out.println( arr.size() );
10
```

Java Interop

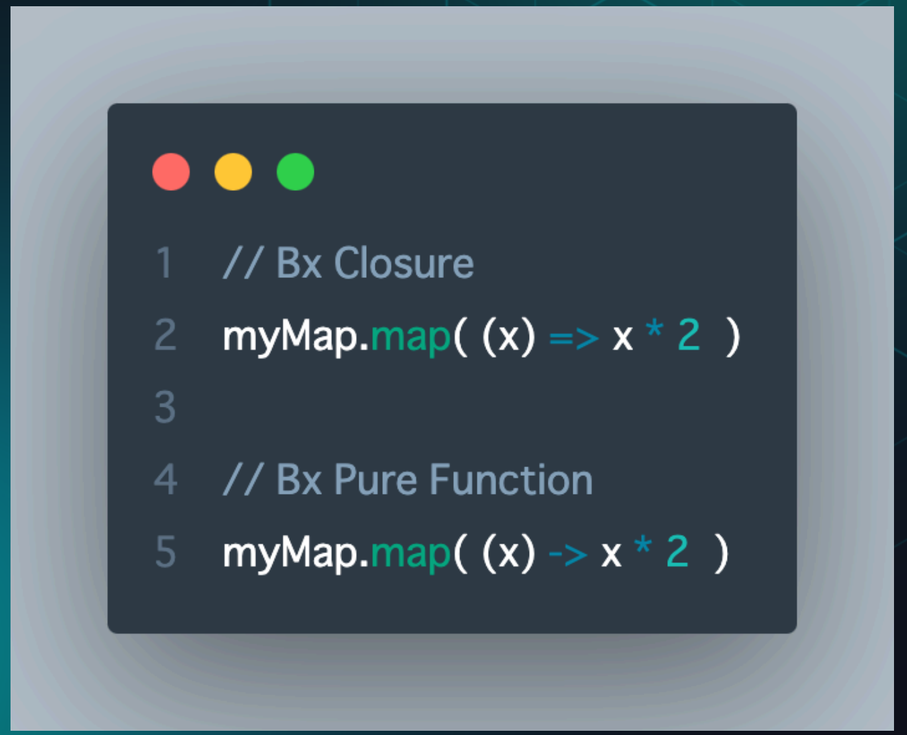
- Concept of object resolvers: java, bx, custom
- New BoxLang Scripting: MyScript.bxs
- Components become Classes: MyClass.bx
- All bx/bxm/bxs are runnable via the OS
- Classes can have a main() runnable convention
- BoxLang annotations



Runnable Classes

Pure Functions + Immutable Classes

- Support not only closures (=>) but pure functions (->)
- No side effects, no carry-over contexts, pure speed
- Especially for asynchronous programming
- New Immutable classes:
  - Arrays, Structs, and Queries
  - Great for async safety, read-only snapshots, speed and more.



Differences between CFML and BoxLang

https://boxlang.ortusbooks.com/getting-started/overview/syntax-style-guide/cfml

Open Your Browser, let's review!

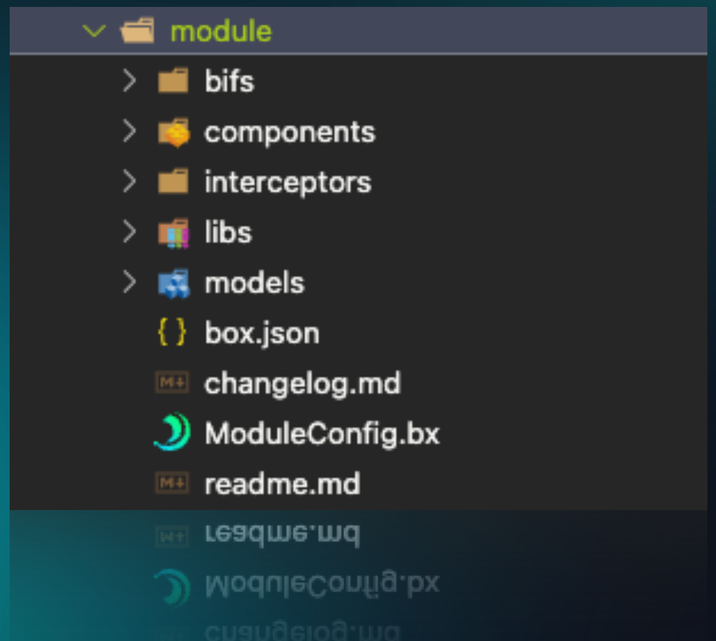
MODULES OVERVIEW

With Jon Clausen

Modular Since Birth

BoxLang Modules

- Inspired by ColdBox modules, NOT OSGI
- Core Runtime with lightest possible footprint
- Taps into the language life-cycle
- Write them in Java or BoxLang or Both!
- Executable as CLI packages
- Integrates with Maven/Gradle



https://github.com/ortus-boxlang/boxlang-module-template

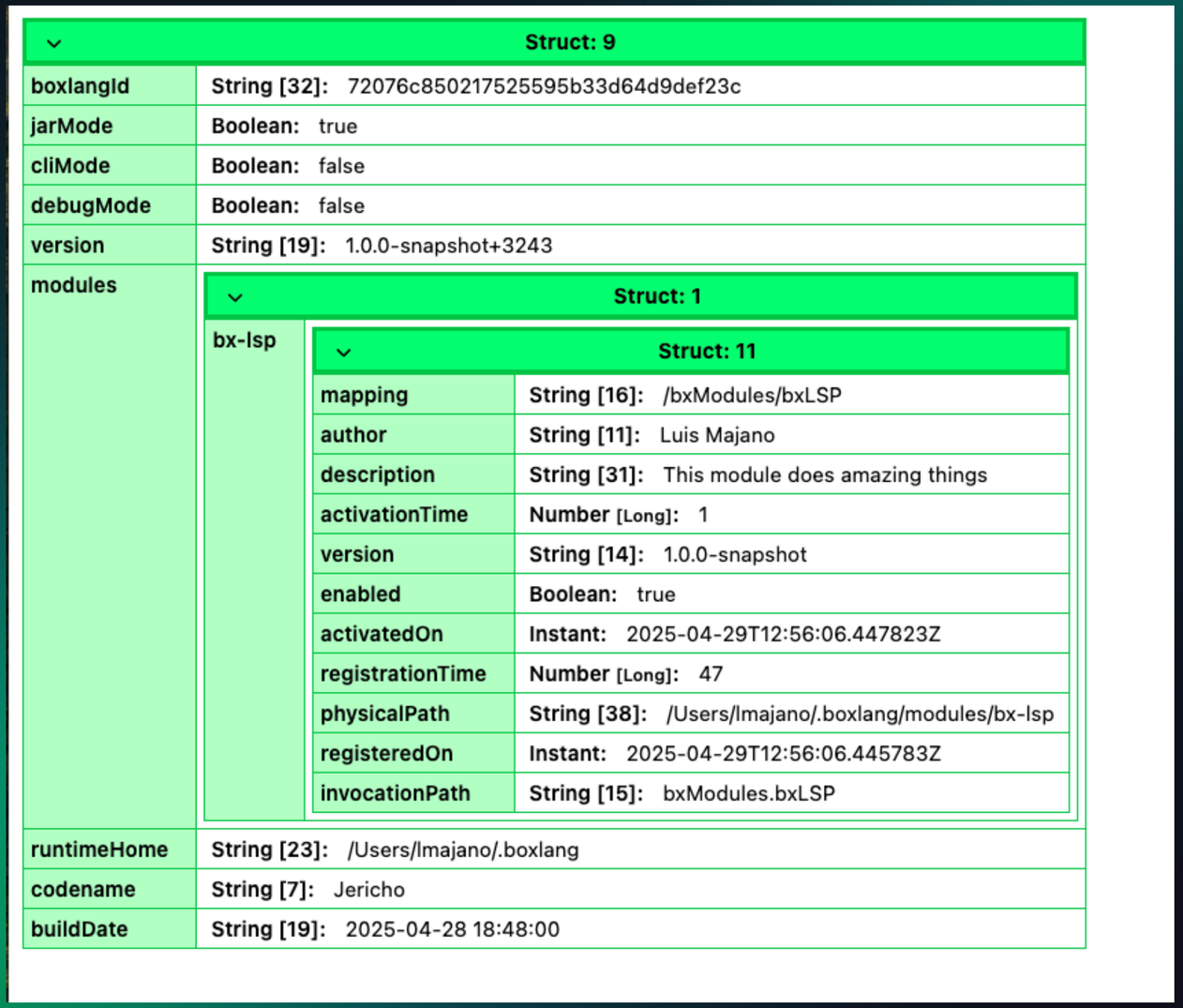
BoxLang CLI Executable Module

- `boxlang module:{name} {args}`
- Executes the `main( args )` method on your ModuleConfig.cfc

```
1  /**
2   * The main method of the module.  This is the entry point of the modu
3   * when it's being executed from the CLI
4   */
5  function main( args = {} ){
6      println( "Executing from the test module" );
7  }
```

BoxLang Runtime Information

- `bx:dump var=#server.boxlang#`
- `getBoxRuntime().getModuleService()`
- BIFS
  - `getModuleList()`
  - `getModuleInfo()`
  - `getFunctionList()`



Struct: 9	
boxlangId	String [32]: 72076c850217525595b33d64d9def23c
jarMode	Boolean: true
cliMode	Boolean: false
debugMode	Boolean: false
version	String [19]: 1.0.0-snapshot+3243
modules	Struct: 1
bx-lsp	Struct: 11
mapping	String [16]: /bxModules/bxLSP
author	String [11]: Luis Majano
description	String [31]: This module does amazing things
activationTime	Number [Long]: 1
version	String [14]: 1.0.0-snapshot
enabled	Boolean: true
activatedOn	Instant: 2025-04-29T12:56:06.447823Z
registrationTime	Number [Long]: 47
physicalPath	String [38]: /Users/lmajano/.boxlang/modules/bx-lsp
registeredOn	Instant: 2025-04-29T12:56:06.445783Z
invocationPath	String [15]: bxModules.bxLSP
runtimeHome	String [23]: /Users/lmajano/.boxlang
codename	String [7]: Jericho
buildDate	String [19]: 2025-04-28 18:48:00

Modules = Runtime Innovation

Open-Source

- CFML
- AI
- CSRF
- ORM
- ESAPI
- FTP
- Image
- Jython
- Mail
- OSHI
- PDF
- Forms
- Markdown
- Evaluate
- WDDX
- YAML
- Password Encryption
- Ini
- Web Support
- ColdBox
- TestBox
- Command Box

JDBC

- Derby
- MySQL
- MariaDB
- SQL Server
- Oracle
- Postgre SQL
- Hypersonic
- Neo4J

+/++

- Redis
- Couchbase
- Mongo
- Elastic
- AI+
- PDF+
- SFTP

BoxLang Modules Installation

- OS binary
  - `install-bx-module {slug}`
- CommandBox
  - `install {slug}`

THANK YOU

Ortus Solutions

BOXLANG®