Article
Eduard Lebedyuk · Jan 26, 2022 4m read

Container configuration management

If you're deploying to more than one environment/region/cloud/customer, you will inevitably encounter the issue of configuration management.

While all (or just several) of your deployments can share the same source code, some parts, such as configuration (settings, passwords) differ from deployment to deployment and must be managed somehow.

In this article, I will try to offer several tips on that topic. This article talks mainly about container deployments.

Codebase unification

Before we get into configuration management, let's talk about codebase unification. The issue is as follows: codebase *must* generally aim to coalesce into one version. Of course, at any point, you'll have several versions of your codebase - DEV version with all-new features, TEST version with additional test code, PROD version, and so on. And that is *fine* because, in this example, changes coalesce into one version over time. Several DEV branches become one TEST branch and so on.

The problem starts when we have several versions at the final step of our deployment pipeline. For example, customer ABC asked for a particular feature XYZ, and now there are two versions of the PROD codebase - with XYZ feature and without it. This is problematic because it immediately doubles our build times and resources consumption. Sure, sometimes there's no way about it. Still, suppose you have such persistent codebase divergencies. In that case, it might be worthwhile to research if you can roll them into one version - in this scenario feature XYZ is activated or not activated on startup.

That said, let's get into configuration management.

Configuration management

What do we want?

- To not store every possible configuration in one container (it's insecure for one, and we still somehow need to make a choice which configuration to apply for another)
- To not build a container for every configuration (simplifies CD pipeline)

We need to pass configuration (settings, secrets, ...) during InterSystems IRIS startup and apply it to our application.

Passing Configuration

There are several ways to pass configuration at startup.

Environment variables

Quick and easy. In case you want to store everything in one variable, use JSON for serialization. Also, remember that Windows has a 32K character limit on environment variables (megabytes on Linux). To retrieve environment variable use $SYSTEM.Util.GetEnviron("ENVVARNAME") . The main advantage is the ease of implementation. Also, check your CD tool - it usually has at least some form of environment variables support during pipeline execution, sometimes with secrets support to make it more secure.

Mounted files

Configuration can be a file mounted on startup. The advantage is no length limits. The disadvantage is that not all cloud offerings support mounted files, and even if they do, file management can be trickier than environment variable management. Be aware that files can always be retrieved from the old layers of a container, even if they were deleted at a later layer.

CPF Merge

System settings can be passed via Configuration Merge File. And its built-in functionality. Disadvantage: CPF Merge File does not deal with application-level settings.

Docker/CSP Secrets

Docker can mount a secret as a file inside a container. Cloud offerings often provide similar functionality. Use it instead of plain files to improve security.

Parsing configuration

Okay, you passed configuration inside a container. What now?

%ZSTART

%ZSTART is your custom code, executed on system start. It looks like this:

```
SYSTEM
    try {
        new $namespace
        set $namespace = "USER"
        // apply configuration
        set $namespace = "%SYS"
    } catch ex {
        zn "%SYS"
        do ex.Log()
    }
    quit 1
```

Important to note that %ZSTART must be tested *thoroughly* - as InterSystems IRIS would fail to start (or behave erratically) if %ZSTART routine returns an error.

Now you're ready to apply your settings.

How you store/apply app-level settings naturally depends on the app, but System Default Settings are available if you use interoperability. Even if you don't use interoperability productions, you can still use System Default Settings as they are available for everyone (as long as interoperability is enabled in a namespace).

To create a new setting, execute:

```
Set setting = ##class(Ens.Config.DefaultSettings).%New()
Set setting.ProductionName = production
Set setting.ItemName = itemname
Set setting.HostClassName = hostclassname
Set setting.SettingName = settingname
Set setting.SettingValue = settingvalue
Set setting.Description = description
Set sc = setting.%Save()
```

And to retrieve setting call either:

```
set sc = ##class(Ens.Config.DefaultSettings).%GetSetting(production, itemname, hostcl
assname, "", settingname, .settingvalue)
set settingvalue = ##class(Ens.Director).GetProductionSettingValue(production, settin
gname, .sc)
```

Interoperability Hosts receive their settings automatically on Business Host job startup.

Summary

Making your application configurable when deploying to more than one server is essential. It can be pretty easy at the early part of the development cycle, but the costs rise with each hardcoded URL. InterSystems IRIS offers several tools you can use to configure the state of your application at startup.

How do you manage multiple deployments?

#Cloud #Containerization #Continuous Delivery #Deployment #InterSystems IRIS

Source URL:https://community.intersystems.com/post/container-configuration-management