Article
Eduard Lebedyuk · Sep 26, 2022 11m read

Continuous Delivery of your InterSystems solution using GitLab - Part XI: Interoperability

Welcome to the next chapter of my CI/CD series, where we discuss possible approaches toward software development with InterSystems technologies and GitLab.

Today, let's talk about interoperability.

Issue

When you have an active interoperability production, you have two separate process flows: a working production that processes messages and a CI/CD process flow that updates code, production configuration and system default settings.

Clearly, CI/CD process affects interoperability. But questions are:

- What exactly happens during an update?
- What do we need to do to minimize or eliminate production downtime during an update?

Terminology

- Business Host (BH) - one configurable element of Interoperability Production: Business Service (BS), Business Process (BP, BPL), or Business Operation (BO).
- Business Host Job (Job) - InterSystems IRIS job that runs Business Host code and is managed by Interoperability production.
- Production - interconnected collection of Business Hosts.
- System Default Settings (SDS) - values that are specific to the environment where InterSystems IRIS is installed.
- Active Message - a request which is currently being processed by one Business Host Job. One Business Host Job can have a maximum of one Active Message. Business Host Job, which does not have an Active Message, is idle.

What's going on?

Let's start with the Production Lifecycle.

Production Start

First of all, Production can be started. Only one production per namespace can run simultaneously, and in general (unless you really know what and why you're doing it), only one production should be run per namespace, ever. Switching back and forth in one namespace between two or more different productions is not recommended. Starting production starts all enabled Business Hosts defined in the production. Failure of some Business Hosts to start does not affect Production start.

Tips:

- Start the production from the System Management Portal or by calling: ##class(Ens.Director).StartProduction("ProductionName")
- Execute arbitrary code on Production start (before any Business Host Job is started) by implementing an OnStart method
- Production start is an auditable event. You can always see who and when did that in the Audit Log.

Production Update

After Production has been started, Ens.Director continuously monitors running production. Two production states exist: *target state*, defined in production class, and System Default Settings; and *running state* - currently running jobs with settings applied when the jobs were created. If desired and current states are identical, everything is good, but production could (and should) be updated if there's a difference. Usually, you see that as a red Update button on the Production Configuration page in the System Management Portal.

Updating production means an attempt to get the current Production state to match the target Production state.

When you run ##class(Ens.Director).UpdateProduction(timeout=10, force=0) To Update the production, it does the following for each Business Host:

1. Compares active settings to production/SDS/class settings
2. If, and only if (1) shows a mismatch, the Business Host would be marked as out-of-date and requiring an update.

After running this for each Business Host, UpdateProduction builds the set of changes:

- Business Hosts to stop
- Business Hosts to start
- Production settings to update

And after that, applies them.

This way, “ updating ” settings without changing anything results in no production downtime.

Tips:

- Update the production from the System Management Portal or by calling: ##class(Ens.Director).UpdateProduction(timeout=10, force=0)
- Default System Management Portal update timeout is 10 seconds. If you know that processing your messages takes more than that, call Ens.Director:UpdateProduction with a larger timeout.
- Update Timeout is a production setting, and you can change it to a larger value. This setting applies to the System Management Portal.

Code Update

UpdateProduction DOES NOT UPDATE the BHs with out-of-date code. This is a safety-oriented behavior, but if you want to automatically update all running BHs if the underlying code changes, follow these steps:

First, load and compile like this:

```
do $system.OBJ.LoadDir(dir, "", .err, 1, .load)
do $system.OBJ.CompileList(load, "curk", .errCompile, .listCompiled)
```

Now, listCompiled would contain all items which were actually compiled (use git diffs to minimize loaded set) due to the u flag. Use this listCompiled to get a $lb of all classes which were compiled:

```
set classList = ""
set class = $o(listCompiled(""))
while class'="" {
  set classList = classList _ $lb($p(class, ".", 1, *-1))
  set class=$o(listCompiled(class))
}
```

And after that, calculate a list of BHs which need a restart:

```
SELECT %DLIST(Name) bhList
FROM Ens_Config.Item
WHERE 1=1
  AND Enabled = 1
  AND Production = :production
  AND ClassName %INLIST :classList
```

Finally, after obtaining bhList stop and start affected hosts:

```
for stop = 1, 0 {
  for i=1:1:$ll(bhList) {
    set host = $lg(bhList, i)
    set sc = ##class(Ens.Director).TempStopConfigItem(host, stop, 0)
  }
  set sc = ##class(Ens.Director).UpdateProduction()
}
```

Production Stop

Productions can be stopped, which means sending a request to all Business Host Jobs to shut down (safely, after they are done with their active messages, if any).

Tips:

- Stop the production from the System Management Portal or by calling: ##class(Ens.Director).StopProduction(timeout=10, force=0)
- Default System Management Portal stop timeout it 120 seconds. If you know that processing your messages takes more than that, call Ens.Director:StopProduction with a larger timeout.
- Shutdown Timeout is a production setting. You can change that to a larger value. This setting applies to the System Management Portal.
- Execute arbitrary code on Production stop by implementing an OnStop method
- Production stop is an auditable event, you can always see who and when did that in the Audit Log.

The important thing here is that Production is a sum total of the Business Hosts:

- Starting production means starting all enabled Business Hosts.
- Stopping production means stopping all running Business Hosts.
- Updating production means calculating a subset of Business Hosts which are out of date, so they are first stopped and immediately after that started again. Additionally, a newly added Business Host is only started,

and a Business Host deleted from production is just stopped.

That brings us to the Business Hosts lifecycle.

Business Host Start

Business Hosts are composed of identical Business Hosts Jobs (according to a Pool Size setting value). Starting a Business Host means starting all Business Hosts Jobs. They are started in parallel.

Individual Business Host Job starts like this:

1. Interoperability JOBs a new process that would become a Business Host Job.
2. The new process registers as an Interoperability job.
3. Business Host code and Adapter code is loaded into process memory.
4. Settings related to a Business Host and Adapter are loaded into memory. The order of precedence is:
   a. Production Settings (overrides System Default and Class Settings).
   b. System Default Settings (overrides Class Settings).
   c. Class Settings.
5. Job is ready and starts accepting messages.

After (4) is done, the Job can ' t change settings or code, so when you import new/same code and new/same systems default settings, it does not affect currently running Interoperability jobs.

Business Host Stop

Stopping a Business Host Job means:

1. Interoperability orders Job to stop accepting any more messages/inputs.
2. If there ' s an active message, Business Host Job has timeout seconds to process it (by completing it - finishing OnMessage method for BO, OnProcessInput for BS, state S<int> method for BPL BPs, and On* method for BPs).
3. If an active message has not been processed till the timeout and force=0, production update fails for that Business Host (and you ' ll see a red Update button in the SMP).
4. Stop succeeds if anything on this list is true:
   - No active message
   - Active message was processed before the timeout
   - Active message was not processed before the timeout BUT force=1
5. Job is deregistered with Interoperability and halts.

Business Host Update

Business host update means stopping currently running Jobs for the Business Host and starting new Jobs.

Business Rules, Routing Rules, and DTLs

All Business Hosts immediately start using new versions of Business Rules, Routing Rules, and DTLs as they become available. A restart of a Business Host is not required in this situation.

Offline updates

Sometimes, however, Production updates require downtime of individual Business Hosts.

Rules depend on new code

Consider the situation. You have a current Routing Rule X which routes messages to either Business Process A or B based on arbitrary criteria.
In a new commit, you add, simultaneously:

- Business Process C
- A new version of Routing Rule X, which routes messages to A, B, or C.

In this scenario, you can't just load the rule first and update the production second. Because the newly compiled rule would immediately start routing messages to Business Process C, which InterSystems IRIS might not have yet compiled, or Interoperability did not yet Update to use.
In this case, you need to disable the Business Host with a Routing Rule, update the code, update production and enable the Business Host again.

Notes:

- If you update a production using produciton deployment file it would automatically disable/enable all affected BHs.
- For InProc invoked hosts, the compilation invalidates the cache of the particular host held by the caller.

Dependencies between Business Hosts

Dependencies between Business Hosts are critical. Imagine you have Business Processes A and B, where A sends messages to B.
In a new commit, you add, simultaneously:

- A new version of Process A, which sets a new property X in a request to B
- A new version of Process B which can process a new property X

In this scenario, we MUST update Process B first and A second. You can do this in one of two ways:

- Disable Business Hosts for the duration of the update
- Split the update into two: first, update Process B only, and after that, in a separate update, start sending messages to it from Process A.

A more challenging variation on this theme, where new versions of Processes A and B are incompatible with old versions, requires Business Host downtime.

Queues

If you know that after the update, a Business Host will not be able to process old messages, you need to guarantee that the Business Host Queue is empty before the update. To do that, disable all Business Hosts that send messages to the Business Host and wait till its queue becomes empty.

State change in BPL Business Processes

First, a little intro into how BPL BPs work. After you compile a BPL BP, two classes get created into the package with the same name as a full BPL class name:

- Thread1 class contains methods S1, S2, ... SN, which correspond to activities within BPL

- Context class has all context variables and also the next state which BPL would execute (i.e., S5)

Also BPL class is persistent and stores requests currently being processed.

BPL works by executing S methods in a Thread class and correspondingly updating the BPL class table, Context table, and Thread1 table where one message "being processed" is one row in a BPL table. After the request is processed, BPL deletes the BPL, Context, and Thread entries. Since BPL BPs are asynchronous, one BPL job can simultaneously process many requests by saving information between S calls and switching between different requests.
For example, BPL processed one request till it got to a sync activity - waiting for an answer from BO. It would save the current context to disk, with %NextState property (in Thread1 class) set to response activity S method, and work on other requests until BO answers. After BO answers, BPL would load Context into memory and execute the method corresponding to a state saved in %NextState property.

Now, what happens when we update the BPL?
First, we need to check that at least one of the two conditions is satisfied:

- During the update, the Context table is empty, meaning no active messages are being worked on.
- The New States are the same as the old States, or new States are added after the old States.

If at least one condition is satisfied, we are good to go. There are either no pre-update requests for post-update BPL to process, or States are added at the end, meaning old requests can also go there (assuming that pre-update requests are compatible with post-update BPL activities and processing).

But what if you have active requests in processing and BPL changes state order? Ideally, if you can wait, disable BPL callers and wait till the Queue is empty. Validate that the Context table is also empty. Remember that the Queue shows only unprocessed requests, and the Context table stores requests which are being worked on, so you can have a situation where a very busy BPL shows zero Queue size, and that's normal.
After that, disable the BPL, perform the update and enable all previously disabled Business Hosts.

If that's not possible (usually in a case where there is a very long BPL, i.e., I remember updating one that took around a week to process a request, or the update window is too short), use BPL versioning.

Alternatively, you can write an update script. In this update script, map old next states to new next states and run it on Thread1 table so that updated BPL can process old requests. BPL, of course, must be disabled for the duration of the update.
That said, it's an extremely rare situation, and usually, you don't have to do this, but if you ever need to do that, that's how.

Conclusion

Interoperability implements a sophisticated algorithm to minimize the number of actions required to actualize Production after the underlying code change. Call UpdateProduction with a safe timeout on every SDS update. For every code update, you need to decide on an update strategy.

Minimizing the amount of compiled code by using git diffs helps with the compilation time, but "updating" the code with itself and recompiling it or "updating" the settings with the same values does not trigger or require a Production Update.

Updating and compiling Business Rules, Routing Rules, and DTLs makes them immediately accessible without a Production Update.

Finally, Production Update is a safe operation and usually does not require downtime.

Links

- Ens.Director
- Building git diffs
- System Default Settings

Author would like to thank @James MacKeith, @Dmitry Zasypkin, and @Regilo Regilio Guedes de Souza for their invaluable help with this article.

#Business Process (BPL) #Continuous Delivery #Continuous Integration #Git #Interoperability #InterSystems IRIS
