Practical CI/CD for ABAP

Lars Hvam, Heliconia Labs, June 2021

Home	https://github.com/heliconialabs/practical-ci-cd-for-abap
License	Creative Commons Attribution 4.0 International
Build	2021-09-24 10:15:23 UTC

1. Introduction

Basic CI/CD should be the default for ABAP development, not the exception. This document introduces a incremental CI/CD setup for releases 702 and up.

The abapGit project uses abaplint to check for syntax errors before merges, this helps ensuring a stable `main` branch. A similar, but more complex, approach can be used to perform validations in classic SAP landscapes where changes are deployed using CTS.

As ABAP is a complicated language, occasionally syntax errors make it into the `main` branch, due to missing checks in abaplint. The bug is then fixed in abaplint, and similar errors can henceforth be detected.

1.1. CTS

CTS will be used for deploying changes between systems in the local landscape.

See the CTS is beautiful blog for more insights.

1.2. Open Source Tooling

The approach is based on open source tooling, this allows developers to share fixes, and promotes an open culture across organizations.

- abapGit
- abaplint

The tooling is delivered independently of SAP releases(702+).

And issues can be reproduced on a local PC by developers, on normal sized hardware.

1.3. CI and CD differences

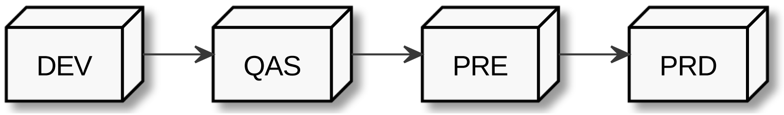
The Continuous Integration and Continuous Delivery terms have different interpretations, but its important to note some of the conceptual differences:

Table 1. CI/CD concepts

Continuous Integration	Continuous Delivery
Parallel	Sequential
Speed	Safety
Flexibility	Auditable
Merge	Overwrite
Style and Syntax	Syntax Only
Per commit	Per build artifact(Transport)
Before code review	After code review

1.4. Landscape

The setup will take offset in a classic 4 tier system landscape,



`DEV`: Source for all CTS transports

`QAS`: User acceptance test by business users

`PRE`: Important, the development artifacts in this system matches PRD. Transports are imported to this system just before PRD, in case there is a syntax error the change is abandoned. This will secure no syntax errors in PRD after import.

The scenario can be expanded with more system tiers and transport routes.

2. CI

Use abaplint for CI, if developing in a de-central git focused setup abaplint.app is one click installation for code review and CI.

For central development, abap-code-review-guide.pdf describes various approaches, which enables code review and CI.

3. CD

The goal is to avoid syntax errors when importing workbench transports, this is done by storing objects from transports and systems in git, then simulating the state after import and verifying by running abaplint.

The user will not be allowed to release a transport from DEV if it results in syntax errors in QAS. Similarly, the setup will give a list of transports which are good to import into follow-on systems.

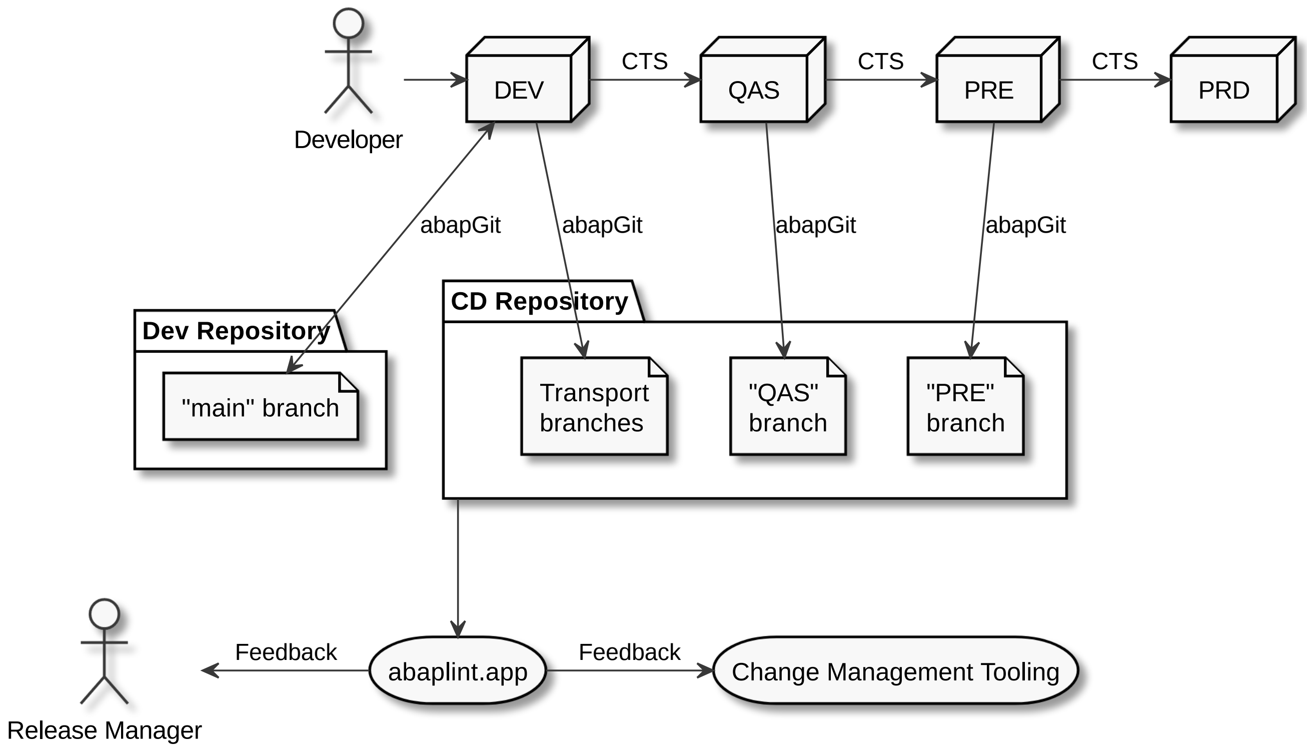
The result of this setup are simple lists of good transports, but there are multiple automatic processes running to produce the lists, this chapter describes the proposed setup.

3.1. Git Mirroring

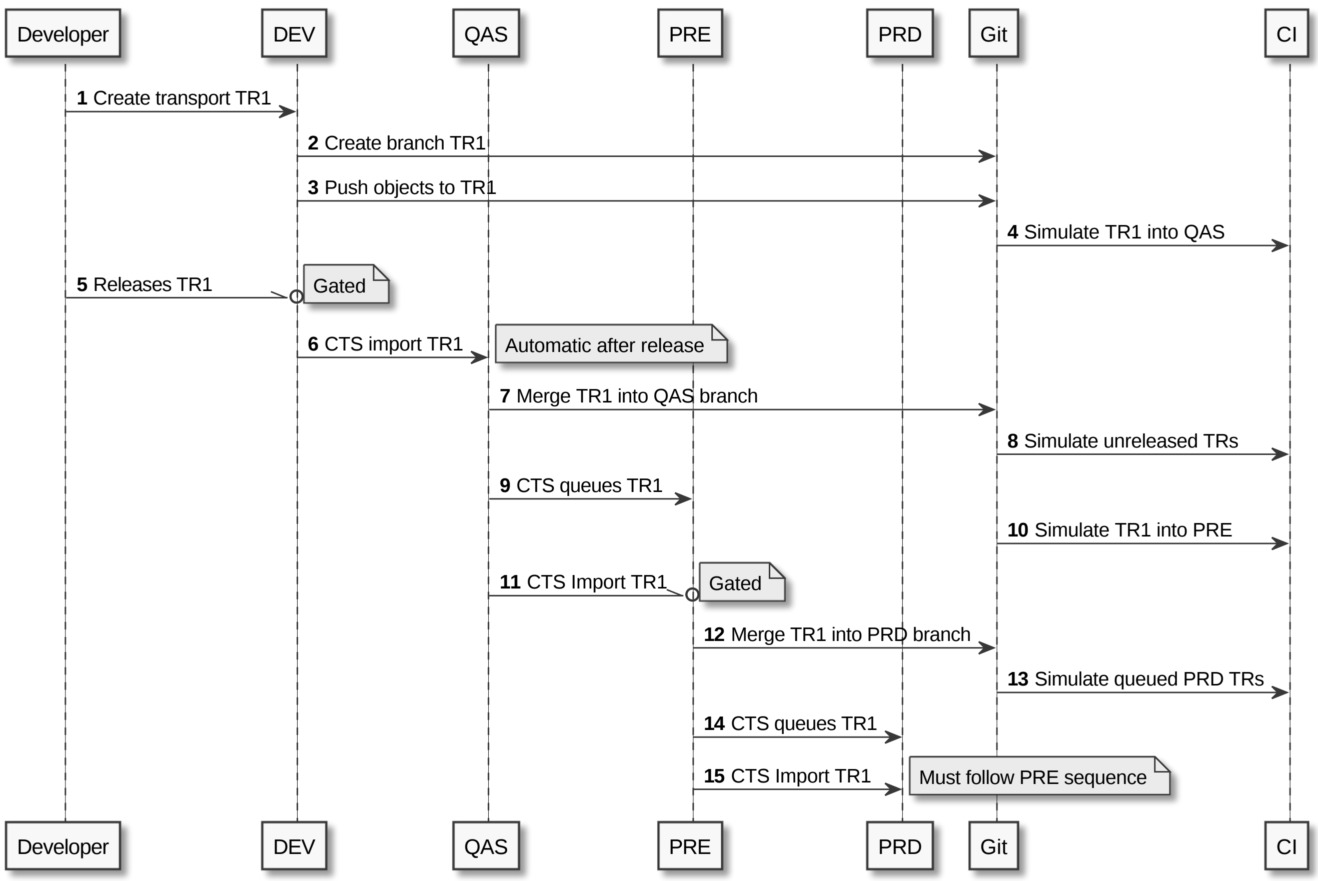
To support a CD setup, systems are mirrored into a specific CD repository, developers will not be working with this repository, it is only used for automation. No actual git features are used, git is used only for storing files.

Transports from DEV are mirrored into branches, the objects of the QAS system are mirrored into a QAS branch and PRE into a PRE branch. PRD is not mirrored, the CD pipeline is executed before PRD.

Developers can use their favorite tooling to do changes in DEV, these changes are recorded into transport requests.



Below diagram gives an example of the lifecycle for a single transport request "TR1":



This is the simplest scenario, transport of copies and more system tiers can be added or removed.

3.2. Maintenance Tasks

Few maintenance tasks to perform regularly,

1. Update abapGit on DEV+QAS+PRE(PRD not needed)
2. Check QAS and PRD branches are in sync with systems
3. Keeping abaplint configuration up to date

3.3. Prerequisites

1. Force independent transports at R3TR level, this can be done with some simple reporting

3.4. Open Questions

1. Exactly how to store things ⇒ manageable
2. Rollback of ABAP objects, out of scope, abapGit solution
3. SAP notes causing syntax errors in custom code, out of scope

3.5. Feasibility

The CD setup will require many jobs to run on the CI servers. This section gives some overall estimates on the number of CI runs given a number of transport requests.

3.5.1. Sequential queue clearing

Importing a queue of 10 transport requests, one by one in sequence. Below numbers are for one system, but it must be run for both QAS and PRE in the example landscape.

1. Import TR1, trigger CI on "main", rebase 9 branches
2. Import TR2, trigger CI on "main", rebase 8 branches
3. Import TR3, trigger CI on "main", rebase 7 branches
4. Import TR4, trigger CI on "main", rebase 6 branches
5. Import TR5, trigger CI on "main", rebase 5 branches
6. Import TR6, trigger CI on "main", rebase 4 branches
7. Import TR7, trigger CI on "main", rebase 3 branches
8. Import TR8, trigger CI on "main", rebase 2 branches
9. Import TR9, trigger CI on "main", rebase 1 branches
10. Import TR10, trigger CI on "main"

sequential(10) = 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 55

$$sequential(n) = \frac{1}{2}n(n+1)$$

https://en.wikipedia.org/wiki/Triangular_number

Table 2. CI runs per TR queue

TRs	CI Runs
10	55
20	210
50	1275
100	5050

Moving fast in a large SAP installation, having 100 transports per week is realistic(?)

Table 3. Estimate, cost for 5050 CI runs

Scenario	Time per run	Memory	Price/hour	Total
Full container	1 hour	256 gb	USD 3.0	USD 15150
abaplint	5 minutes	8 gb	USD 0.1	USD 42

abaplint 2.74.23, 4 million lines of ABAP and other ABAP artifacts, syntax only, 8 gb memory allocated to node, single core boost = 2.4ghz, in ~10 minutes (`find -name '*.abap' | xargs cat | wc -l`)

Public abaplint-app-performance example, 500k lines of ABAP in 50 seconds.

abaplint does require more work to do proper syntax checks, which will result in slower performance. But, there are also multiple opportunities for optimizations, and abaplint currently runs only single-threaded.

3.5.2. Speculative combinatorics

Example, 3 transports in queue, gives following combinations, note that the sequence is defined by the queue,

Table 4. Import combinations

TR1 TR2 TR3
TR1 TR2
TR1 TR3
TR2 TR3
TR1
TR2
TR3

Binomial coefficient series

$$speculative(n) = 2^n - 1$$

https://en.wikipedia.org/wiki/Binomial_coefficient

Table 5. Speculative combinations

TRs	Calculation	Result
3 TRs	2^3 - 1	7 combinations
4 TRs	2^4 - 1	15 combinations
10 TRs	2^10 - 1	1023 combinations

Assuming full coverage.

Speculative merge results must be configured for a specific maximum depth.

4. Conclusion

CI/CD can be added in existing landscapes with small impact on processes and infrastructure.

1. No new infrastructure (or limited to the PRE system), cloud solution
2. 702 and up
3. Incremental cloud style evolution
4. Self and independently phased CI/CD rollout and upgrade

Secure, only CTS import

6. Secure, only push from DEV,QAS,PRE systems
7. Business continuity, fallback to CTS if git is down
8. Not just referential checks, as things move faster, more exotic scenarios will occur
9. Keep development tooling separate from kernel