CI/CD WORKFLOWS

Mehmet Pekmezci

CI/CD WORKFLOWS

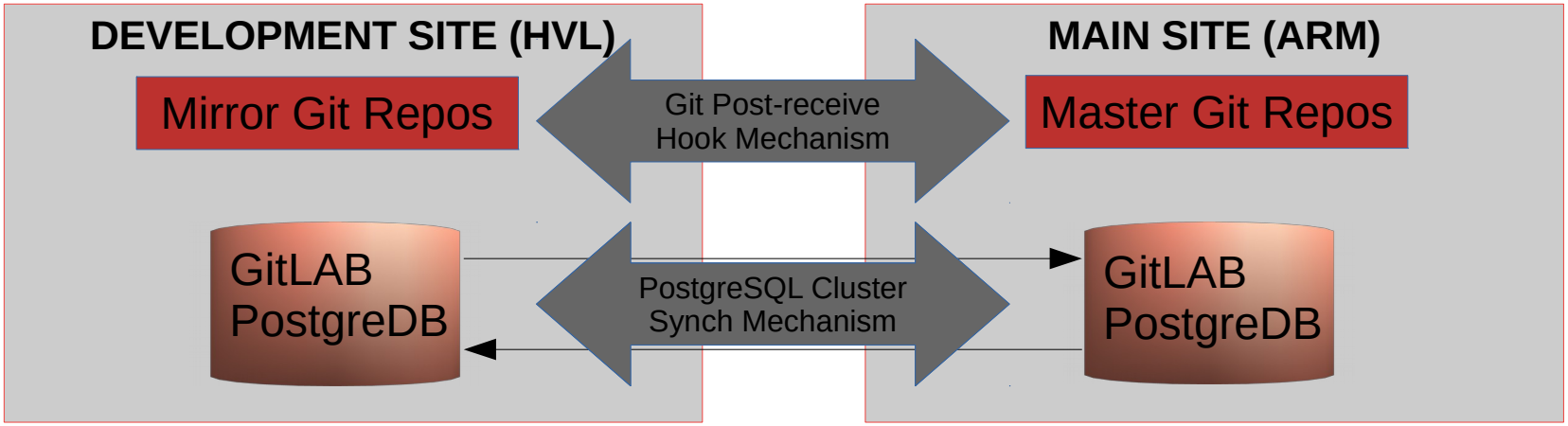
- ENVIRONMENT SETUP
- CSU RELEASE WORKFLOWS
- CONFIGURATION DATABASE
- SDK RELEASE WORKFLOW
- PLATFORMS RELEASE WORKFLOW

ENVIRONMENT SETUP

- GIT CLUSTER as SCM Tool
- MYSQL CLUSTER as Platform Configuration Database
- GitLAB and JIRA as Collaboration Tool.
  - Postgresql Database Cluster for Gitlab
- Jenkins as Build Tool
- Nexus as Artifact Repository
- Ansible as Deploy Tool

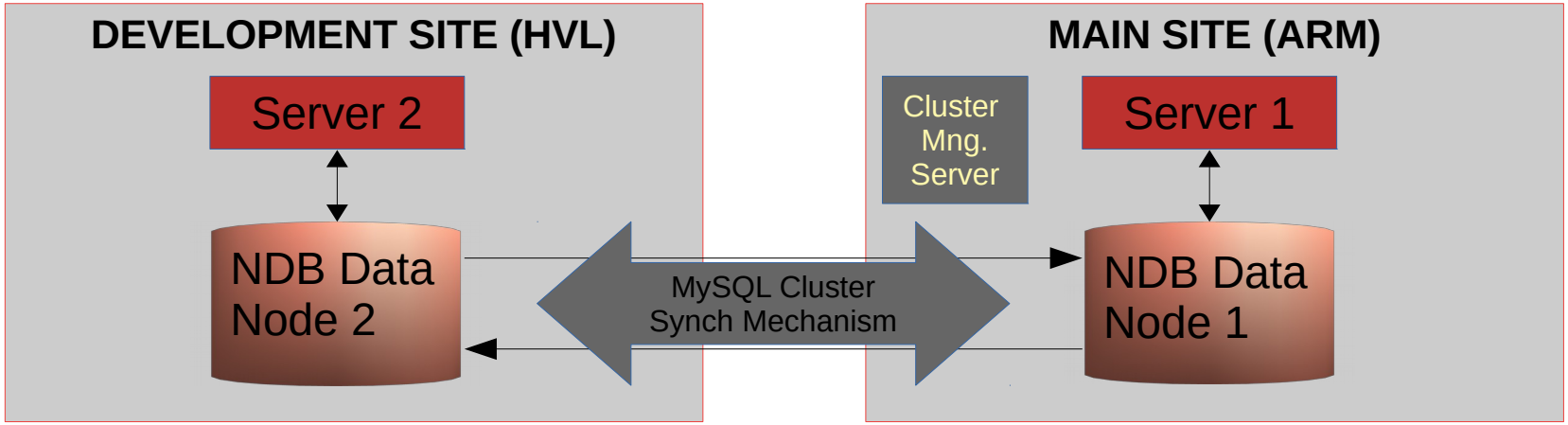
ENVIRONMENT SETUP
SCM CLUSTER

- GIT will be used as SCM Tool.
- All repositories will be mirrored using post-receive hooks
- GitLAB and JIRA will be used as collaboration tool.
- Postgresql database for Gitlab will be clustered



ENVIRONMENT SETUP
CONFIGURATION DATABASE CLUSTER

- Configuration Database will be implemented using MySQL NDB Cluster
- The only thing to be pay attention is make sure the cluster nodes always communicates through network, to prevent split brain problem.



CSU RELEASE WORKFLOWS

- Release workflows of a single CSU (Configurational Software Unit)
- Explained using GIT branches.

  - STANDARD : **“trunk”, “test”, “master”, ”tag”**
  - FIX : **“fix” , “FixTest”, “FixMaster”, “tag”**
  - FEATURE : **“feature”, “FeatureTest”**

CSU STANDARD RELEASE WORKFLOW

1 - Trunk GIT branch

Developer
- commits to this branch
  - multiple times commit
- starts **PreTest** jenkins job
  - **params** : project, platform, sdk_version,
    target_version (3 digits) ,
    related_issue_numbers

Jenkins PreTest job
- update version file, compile, code quality, unit & integration test
- commit version file and create local rpm
- if there exists automated scenario tests
  - deploy to related (project-platform-sdk_ver) dev-test servers
  - run automated scenario tests
  - merge directly trunk to test
- else
  - create new Merge-Request (Pull-Request in gitlab)
  - mail to all developers in the same team.

2 - Test GIT branch

Reviewer (Any other developer in the same team)
- review the code
- merge trunk to test branch (using gitlab web gui)

Jenkins Test job
- change on test branch triggers “Jenkins Test Job”
- compile, create package **version.rc.NO**, upload to nexus
- deploy to related (project-platform-sdk_version) **TTE** (Small Test) environments (lookup from database **free** env.)
- create new Merge-Request (test to master)
- email to test engineer group

Test Engineer
- run tests (may run automated tests) on **TTE**
- merge test to master branch

3 - Master and Tag GIT branch

Jenkins Release job
- change on master branch triggers “Jenkins Release Job”
- if there is already a package in nexus, directly return success. (project-platform-module-version-os-arch-sdk_version)
- if tag of that version not exists (same version for another platform) then create new **tag** with new version (indicated in version file)
- compile the related **tag**
- create package and upload to nexus with new version
- deploy to related(project-platform-sdk_ver) **free** **TTE** environments.
- remotely set related issues to to state **“resolved”**
- if platform == sdk send mail to admin group

CSU FIX RELEASE WORKFLOW

1 - Fix GIT branch

Administrator
- commits related tag to the “fix” branch

Developer
- commits to this branch
  - multiple times commit
- starts **FixPreTest** jenkins job
  - **params** : project, platform, sdk_version,
    target_version (4 digits),
    related_issue_numbers

Jenkins FixPreTest job
- update version file, compile, code quality, unit & integration test
- commit version file and create local rpm
- if there exists automated scenario tests
  - deploy to related (project-platform-sdk_ver) dev-test servers
  - run automated scenario tests
- create new Merge-Request (Pull-Request in gitlab)
- mail to all developers in the same team.

2 - FixTest GIT branch

Reviewer (Any other developer in the same team)
- review the code
- **checks if the fix is covered also in the trunk code**.
- merge **fix** to **FixTest** branch (using gitlab web gui)

Jenkins FixTest job
- change on **FixTest** branch triggers “Jenkins FixTest Job”
- compile, create package **version.rc.NO**, upload to nexus
- deploy to related (project-platform-sdk_version) **TTE** environments
- create new Merge-Request (**FixTest** to **FixMaster**)
- email to test engineer group

Test Engineer
- run tests (may run automated tests) on **TTE**
- merge **FixTest** to **FixMaster** branch

3 - FixMaster and Tag GIT branch

Jenkins FixRelease job
- change on **FixMaster** branch triggers “Jenkins Release Job”
- create new **tag** with new version (indicated in PreTest Job)
- compile the related **tag**
- create package and upload to nexus with new version
- deploy to related(project-platform-sdk_ver) **TTE** environments.
- remotely set related issues to to state **“resolved”**

CSU FEATURE RELEASE WORKFLOW
(For Long Term Feature Implementations)

1 - Feature GIT branch

Developer
- commits to this branch
  - multiple times commit
- starts **FeaturePreTest** jenkins job
  - **params** : project, platform, sdk_version,
    target_version (3 digits . <feature-name>),
    related_issue_numbers
    branch : **feature**

Jenkins FeaturePreTest job
- update version file, compile, code quality, unit & integration test
- commit version file and create local rpm
- if there exists automated scenario tests
  - deploy to related (project-platform-sdk_ver) dev-test servers
  - run automated scenario tests
  - merge directly **feature** to **FeatureTest**
- create new Merge-Request (Pull-Request in gitlab)
- mail to all developers in the same team.

2 - FeatureTest GIT branch

Reviewer (Any other developer in the same team)
- review the code
- merge **feature** to **FeatureTest** branch (using gitlab web gui)

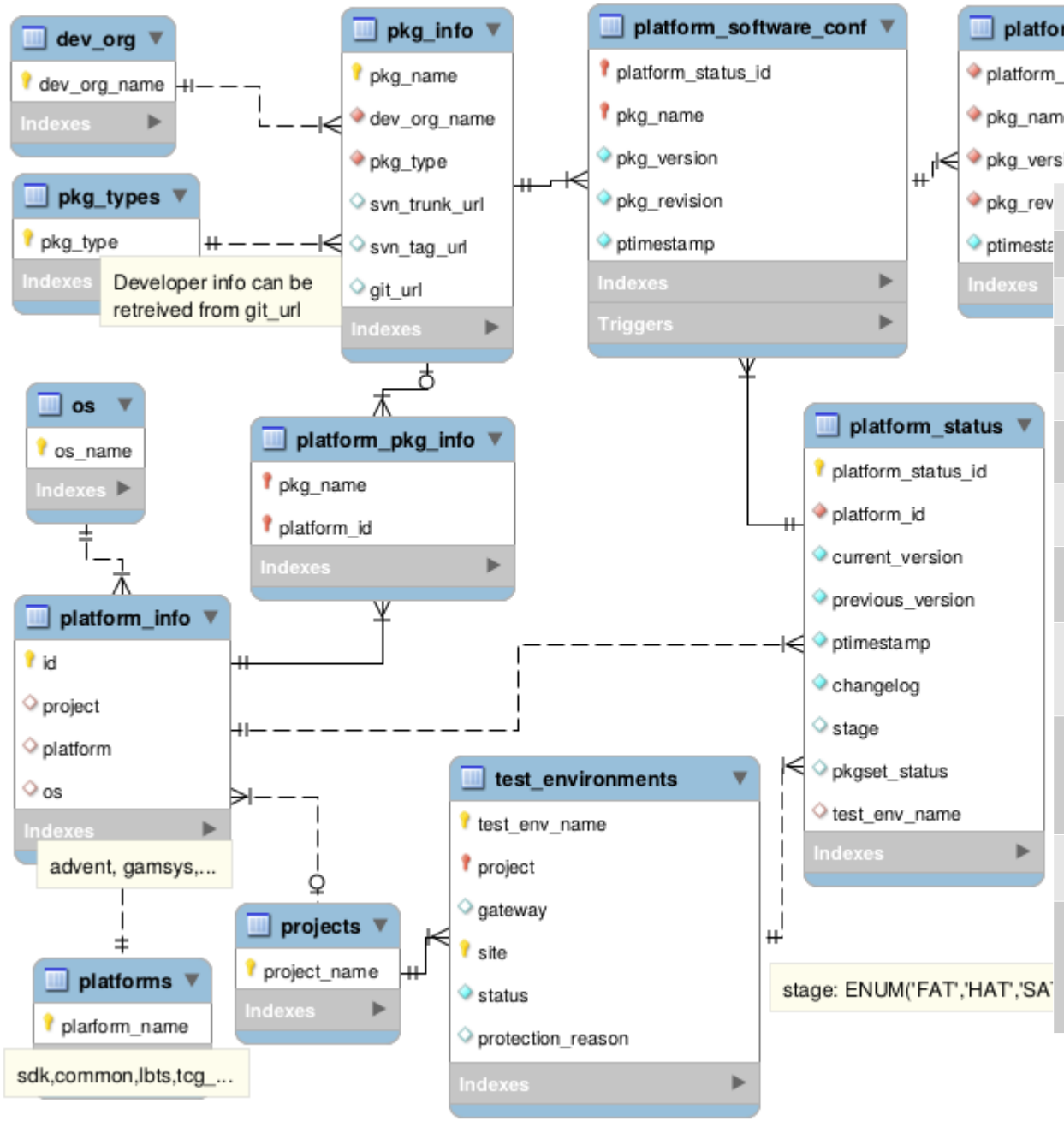
Jenkins FeatureTest job
- change on test branch triggers “Jenkins FeatureTest Job”
- compile, create package **version.rc.NO**, upload to nexus
- deploy to related (project-platform-sdk_version) **TTE** environments
- create new Merge-Request (test to master)
- mail to all developers in the same team.

Developer
- run tests (may run automated tests) on **TTE**
- merge **FeatureTest** to **FeatureMaster** branch

3 - Trunk GIT branch

Senior Developer
- when the long-term feature implementation work is finished, senior developer merges all the code with trunk

CONFIGURATION DATABASE



dev_org	armerkom, havelsan, cots3p
pkg_types	app, infra, gw, cots3p
projects	advent, gamsys, galsys, …
platforms	sdk, lbts, tcg_*
os	CentOS-6.10, CentOS-8.1, …
pkg_info	Software module's name (primary key), organization,type and source code url
platform_info	id is primary key, project, platform, os config. of the platform. (project,platform,os) is unique.
platform_pkg_info	platform's software packages.
platform_status	Current version of the platform. platform_status_id is primary key
platform_software_conf	Software package versions related to a specific version of a platform. platform_status_id, pkg_name is primary key
platform_sotfware_conf_history	Every inserts to platform_software_conf table is logged to this table by a trigger.
test_environments	**name**=TTE1/.. **project**=gamsys/…. **gateway**=ip/hostname **site**=ARM/HVL, **status**=FREE/PROTECTED **protection_reason**="free text"

CONFIGURATION DATABASE ADDING A NEW PLATFORM INFO

1 – Projects Table

Configuration Manager
- if a new project is introduced,
  - starts **CM_Add_Remove_Project** jenkins job
    - **params** : project_name, Add / Remove

Jenkins CM_Add_Remove_Project job
- insert/delete project_name into/from **projects** table
- deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the platform_info table.

2 – Platforms Table

Configuration Manager
- if a new platform is introduced,
  - starts **CM_Add_Remove_Platform** jenkins job
    - **params** : platform_name, Add / Remove

Jenkins CM_Add_Remove_Platform job
- insert/delete platform_name into/from **platforms** table
- deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the platform_info table.

3 – OS Table

Configuration Manager
- if a new os is introduced,
  - starts **CM_Add_Remove_OS** jenkins job
    - **params** : os_name, Add / Remove

Jenkins CM_Add_Remove_OS job
- insert/delete os_name into/from **os** table
- deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the platform_info table.

4 – Platform_Info Table

Configuration Manager
- if a new platform configuration is introduced,
  - starts **CM_Add_Remove_Platform_Info** jenkins job
    - **params** : select project, platform, os , Add / Remove

Jenkins CM_Add_Remove_Platform_Info job
- insert/delete relation (project,platform,os) into/from **platform_info** table
- deleting is safe because of foreign key platform_status table.

5 –Platform_Pkg_Info Table

Configuration Manager
- starts **CM_Platform_Pkg_Info** jenkins job
  - **params** : select platform_info from list, (e.g. advent-sdk-CentOS-8.1)
    select pkg_name check boxes grouped by csci (all is selected by default)

Jenkins CM_Platform_Pkg_Info job
- insert all selected pkg_names with platform_id (deletes all non selected)

CONFIGURATION DATABASE ADDING A NEW SOFTWARE INFO

1 – Dev_Org Table

Configuration Manager
- if a new development organization is introduced,
  - starts **CM_Add_Remove_Dev_Org** jenkins job
    - **params** : dev_org, Add / Remove

Jenkins CM_Add_Remove_Dev_Org job
- insert/delete dev_org into/from **dev_org** table
- deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the pkg_info table.

2 – Pkg_Type Table

Configuration Manager
- if a new software package type is introduced,
  - starts **CM_Add_Remove_Pkg_Type** jenkins job
    - **params** : pkg_type, Add / Remove

Jenkins CM_Add_Remove_Dev_Org job
- insert/delete pkg_type into/from **dev_org** table
- deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the pkg_info table.

3 – Pkg_info Table

Configuration Manager
- starts **CM_Add_Remove_Pkg_Info** jenkins job
  - **params** : pkg_name, Add / Remove, svn_trunk_url, svn_tag_url, git_url

Jenkins CM_Add_Remove_Pkg_Info job
- If Add Selected
  - if git_url is not null
    - remotely add (if not exists) **"gitlab group"** for the corresponding configurational tree branch **(segment/csci/csc)** by parsing url
    - remotely add the repository within the group created above.
  - else if snv_trunk_url and svn_tag_url is not null
    - create the given svn urls using "--parents" parameter.
  - prepare default directory structure
    - copy the template project directory and file structure from the "base/cm/project_template" git/svn repository
    - change release, gradle, gpack.xml files accordingly
    - insert the values into pkg_info table.
- else if remove is selected
  - remove svn/git repositories if exists.
  - deleting is safe because ON DELETE RESTRICT foreign key trigger is defined in the platform_info table.

4 –Platform_Pkg_Info Table

Configuration Manager
- starts **CM_Pkg_Platform_Info** jenkins job
  - **params** : select pkg_name from list, select (project-platform-os) check boxes grouped by project (all is selected by default)

Jenkins CM_Pkg_Platform_Info job
- insert all selected platform_ids with pkg_name (deletes all non selected)

SDK RELEASE WORKFLOW

- Release workflow of a SDK (Software Development Kit)
- Explained using Configuration Database

SDK RELEASE WORKFLOW

1- Platform_Status and Platform_Software_Conf Table

Configuration Manager
- starts **SDK_RELEASE** jenkins job
  - **params** : select (project-platform-os) (only advent-sdk-* will be listed),
    select stage, input changelog, input version
    (version number 4 digit = 3 digit SDK + 1 digit ”0” to indicate SDK)

Jenkins SDK_RELEASE job
- insert/update the values to platform_status table
- insert/remove/update the values to platform_software_conf table (pkg_name and version)
- if this is the first definition of the sdk version
  set pkgset_status to ‘DEFINED’ in the platform_software_conf table.
  else
  set pkgset_status to ‘UPDATED’ in the platform_software_conf table.
- trigger jenkins **SDK_BUILD** job

Jenkins SDK_BUILD job
- this is an explicitly triggable , nightly build job.
- default parameters :
  last sdk version and pkg versions are selected from the platform_software_conf table
- if pkgset_status is ‘BUILT’ or ‘UPDATE_BUILT’ , do nothing return successful.
- create a new docker using base sdk image, login to that image
- initiate release file of that image using sdk version and list of packages in platform_software_conf .
- for each pkg_name-version tag
  - start the jenkins “<pkg_name>_release” job
  - install the sdk rpm package from nexus
- if all builds are successful , create VDD and send mail to admin-mail-group and,
  If pkgset_status is ‘DEFINED’ : set pkgset_status to ‘BUILT’ in the platform_software_conf table.
  else if pkgset_status is ‘UPDATED’ : set pkgset_status to ‘UPDATE_BUILT’ in the platform_software_conf table.
- else
  send mail to admin-mail-group and to failed packages’ all committers (retrieved from git_url).
  Mail notice : Select this new SDK version while building with jenkins.
  “<pkg_name>_release” job will send success message to admin-mail-group
- copy the docker image to the selected OS’s SDK share machine.
  Run the docker image on that machine and mount the image’s SDK dir to machine’s SDK dir by sdk version number
  (e.g. mount image:/*opt*/sdk /opt/sdk/1.0.1.0)

PLATFORMS RELEASE WORKFLOW

- Release workflow of platforms (Test systems and Real systems)
- Explained using Configuration Database

PLATFORMS RELEASE WORKFLOW

1- Platform_Status and Platform_Software_Conf Table

Configuration Manager
- starts **PLATFORM_RELEASE** jenkins job
  - **params** : select (project-platform-os),
    select stage, input changelog, input version
    (version number 4 digit = 3 digit SDK + 1 digit)

Jenkins PLATFORM_RELEASE job
- insert/update the values to platform_status table
- insert/remove/update the values to platform_software_conf table (pkg_name and version)
- if this is the first definition of the sdk version
  set pkgset_status to 'DEFINED' in the platform_status table.
  else
  set pkgset_status to 'UPDATED' in the platform_status table.
- trigger jenkins **PLATFORM_BUILD** job

Jenkins PLATFORM_BUILD job
- this is an explicitly triggable, parallel running (running on different jenkins slaves grouped by csci), nightly build job.
- default parameters :
  last sdk version and pkg versions are selected from the platform_software_conf table
- if pkgset_status is 'BUILT' or 'UPDATE_BUILT' , do nothing return successful.
- initiate related nexus directories.
- for each pkg_name-version tag
  - group by csci
  - start the jenkins "<pkg_name>_release" job
  - install the sdk rpm package from nexus
- if all builds are successful , create VDD and send mail to admin-mail-group and,
  If pkgset_status is 'DEFINED' : set pkgset_status to 'BUILT' in the platform_status table.
  else if pkgset_status is 'UPDATED' : set pkgset_status to 'UPDATE_BUILT' in the platform_status table.
- else
  send mail to admin-mail-group and to failed packages' all committers (retrieved from git_url).
  Mail notice : Select this new SDK version while building with jenkins.
  "<pkg_name>_release" job will send success message to admin-mail-group

PLATFORMS RELEASE WORKFLOW

2- Test_Environments Table

Configuration Manager
- starts **PLATFORM_CONFIGURE_TEST_ENV** jenkins job
  - **params** : select project, select site,
    Input test_env_name, input gateway
    select status, input protection_reason

Jenkins PLATFORM_CONFIGURE_TEST_ENV job
- insert/delete/update the values to test_environments table

3- Deploy to Test Platforms Job

Configuration Manager
- starts **PLATFORM_TEST_DEPLOY** jenkins job
  - **params** : selects (project-platform-os-version),
    select check boxes of project's free test platforms (lookup from database)

Jenkins PLATFORM_TEST_DEPLOY job
- update the selected test_environments with the selected package set.
  - check ansbile_config rpm package's scripts contains all the packages listed in the platform_software_conf table
  - update gsu repo with nexus. (installation server's rpm repository)
  - update ards_core/ansible_config/dns_config in the gsu (installer server's scripts)
  - apply environment specific rules in the ansible_config.
  - check the packages listed in the ansbile_config files are present in the repository.
  - check rpm dependencies of all packages
  - check infogram message consistency of all packages
  - start installation.
  - run base environment health check scripts (ansible_config)and prepare a report (basic services)
  - mail the result (success/failure) and the test report to the admin group
  - trigger **PLATFORM_TEST** jenkins job

PLATFORMS RELEASE WORKFLOW

4- PLATFORM_TEST Jenkins Job

Configuration Manager
- starts **PLATFORM_TEST** jenkins job
    - **params** : select project-site-test_env_name

Jenkins PLATFORM_TEST job
- connect to the gsu (general support unit) of the test environment
- trigger $TEST_BASE_DIR/start.sh
- send the test results to the admin-mail-gorup , test-engineer-mail-group, technical-managers-mail-group