



# CI/CD WORKFLOWS

Mehmet Pekmezci



# CI/CD WORKFLOWS

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



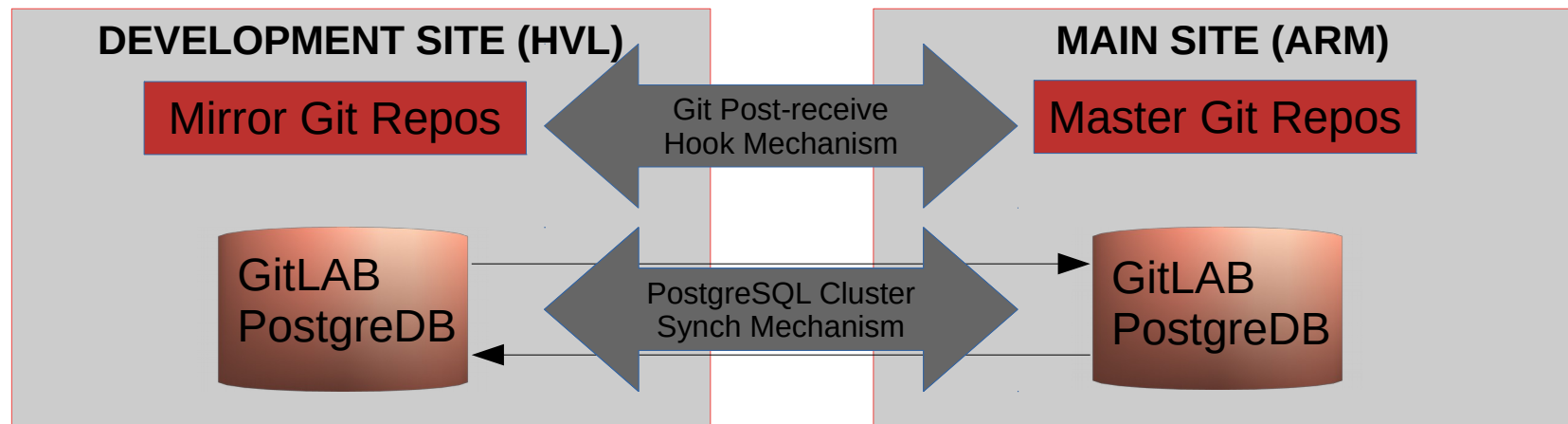
# ENVIRONMENT SETUP

- 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

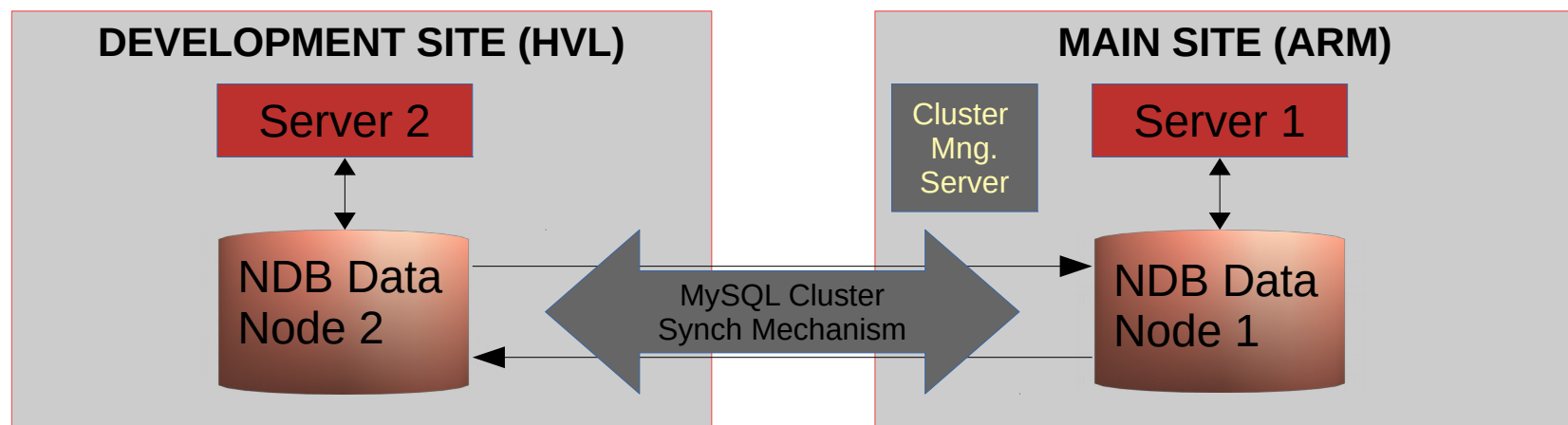
- GIT will be used as SCM Tool.
- All repositories will be mirrored using post-receive hooks
- GitLAB and JIRA will be used as collaboration tool.
- Postgresql database for Gitlab will be clustered



# 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 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

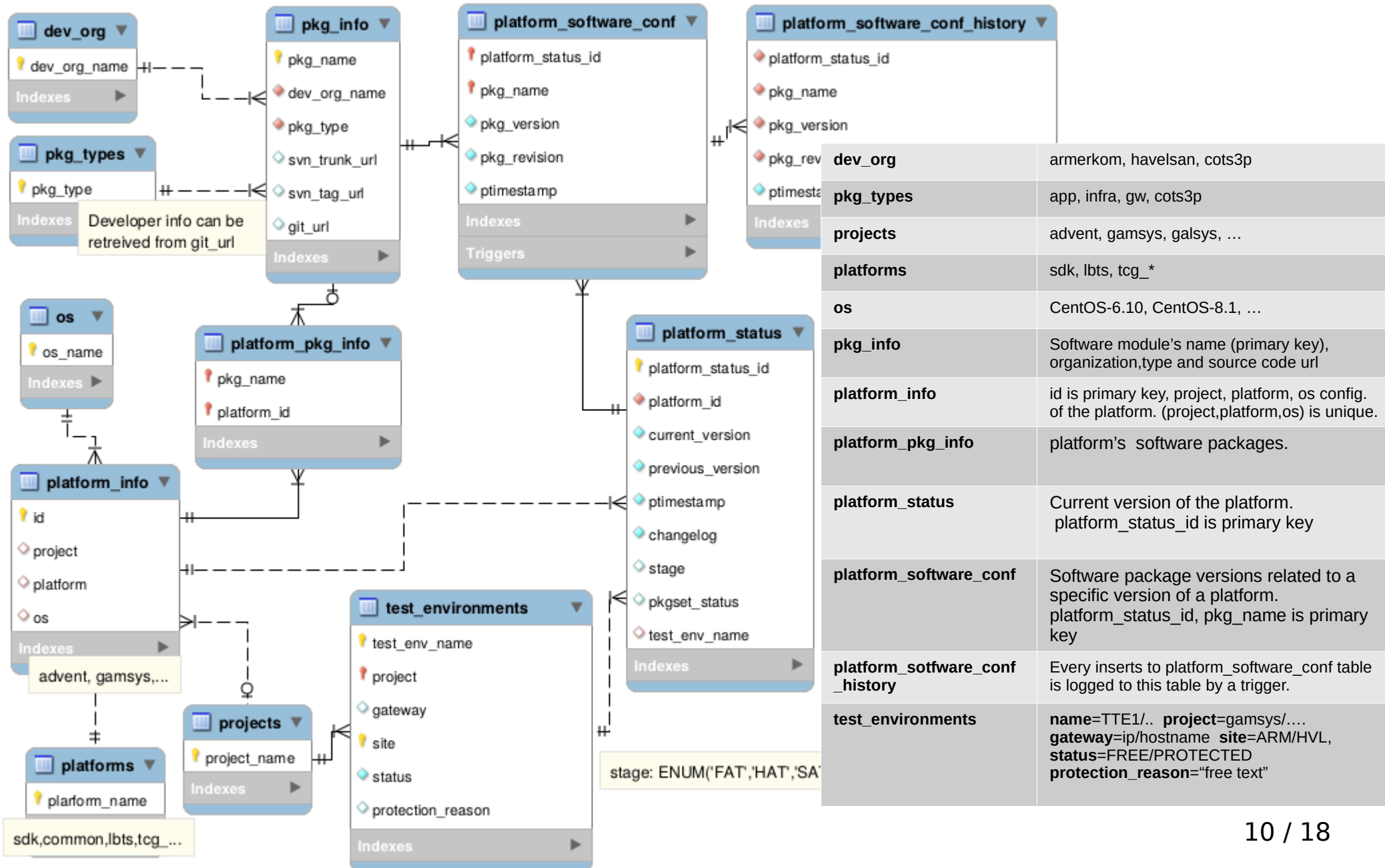
- run tests (may run automated tests) on **TTE**
- merge **FeatureTest** to **FeatureMaster** branch

### 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



# 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 ansible\_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 ansible\_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

## 2- 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-group , test-engineer-mail-group, technical-managers-mail-group