Copado Governance and Administration

You will need to define the **Release process** that your team will follow based on the tasks and necessary steps for every stage there might be.

You have to **adapt Copado to the process** you have previously defined, as Copado is a flexible tool and can be adjusted to your business needs.

You will need to **prepare your team by providing training sessions** so that they get the required knowledge and can work faster.

You will have to follow **best practices** to maintain your copado pipelines and configuration parameters.

Governance and Administration Overview :



Defining your release process is the first and one of the most important steps as you will decide the path to accomplish a streamlined process, starting from when the user stories get created until they are released.

You will need to have answers to the following questions:

* What does a user story need before it can be moved to the next environment? Documentation?
* Documentation review?
* Code coverage?
* Peer review?
* Code analysis?
* Deployment validation?
* What does a user story need to be promoted to production?
* UAT testing?
* Does it include testing integrations?
* What about automated testing?
* How will development conflicts be resolved?
* What are the hotfix and sandbox refresh procedures?
* What release maturity level is required at the moment?
* What about in the future?
* Should you consider version control, automated testing, or deployments only?

You can start by getting a graphical overview so that you can better explain it later to your team. You will then continue by documenting the whole process, including the roles and tasks involved.

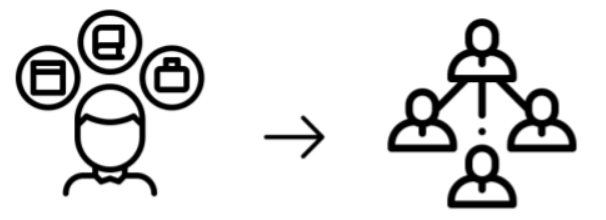
**Define the Release Process**

As mentioned before, as a team, you should follow a process to release features in a controlled way, and that starts by **defining an environment landscape** where your team will work. You will need to answer some questions so that you can then decide what the best approach for your business is, and document it. Here are some of them:

* How many production organizations do you have?
* Are you deploying to multiple production orgs?
* What type of sandboxes do you require? Developer, integration, staging, UAT, hotfix?
* How many developer sandboxes are you going to use? Is it one sandbox for all, one sandbox for each development team or one sandbox for each developer?
* Do you have System Integrators working on specific sandboxes? Are they allowed to deploy to all sandboxes or even to Production?

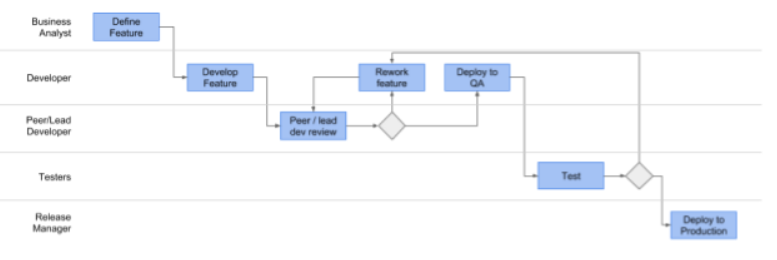
Next, you will need to **define the roles involved** in the release process and the tasks for each of these roles. Teamwork in software development is very important, as multiple parties need to work together to make sure that features are stable and in accordance with end-users expectations. So you need to clarify the roles, their tasks, and what each team member can do to improve the process. Write everything down on a piece of paper or a shared document. You will later have to document it and share it with everyone on the team so that they can access it whenever they need it. Ponder over the following topics and try to clarify all the required details:

* What is owned by functional or business stakeholders? Do they test user stories, report bugs, and submit feature requests? Is there anything else that they should do?
* What steps are owned by developers? Apart from developing, do they do peer review, commit, or deployment validation? Can they deploy to sandboxes but not to production?
* What are the release managers responsible for? Are they the ones who can deploy to production? Are they maintaining orgs in sync, coordinating conflict resolutions and informing stakeholders about upcoming release activities?
* What about QA testers? Are they responsible for performing smoke tests after a deployment and regression tests to ensure existing functionalities are not affected? Is there anything else that they should perform?



By answering these questions and describing the tasks of each role, you will have a better picture of how your permissions will look like beforehand, and ensure that everyone has the proper access. Additionally, any user that is part of your release process can access the documentation and double-check the steps to follow in each stage of the process, if in doubt. This way, you will avoid the lack of discipline in the process that can impact the deployment execution time significantly.

Here is an example of a release process focusing on the tasks per team member:



Let’s have a look at each team member in more detail. You can use these examples as guidance when defining the roles.

**Business Analyst Tasks**:

* Guide the organization towards a reasonable solution and feature iterations.
* Document the solution and feature iterations by providing process context and enough information for the developers to know what to do. This will result in multiple benefits for the team, such as less guesswork during estimations, or creating test scripts in parallel with the development.
* Create functional user stories for the developers to estimate the work.
* Create features that are precise enough to be testable. Who needs to test this feature? Where should they navigate? What actions do they need to perform to get a result?

**Developer Tasks**:

* Provide the technical details of the feature that is being worked on, as it can be crucial for others to understand the overall design in order to make informed decisions when moving the feature towards higher environments.
* Invest time in working according to best practices and testing the work done: write org-agnostic Apex tests, do not hardcode references to a specific org, and perform a validation deployment towards the destination org, including running test classes.
* If working with Version Control (Git), check the feature branch to always make sure that the commits only include changes in the feature that is being worked on. A peer or pull request review process may help identify wrongly committed items.
* Create pre or post-deployment tasks in a comprehensive way so that anyone in the team can perform the task at any time.
* Example process: Create Org Credential for the sandbox the developer is working on, create a user story, commit metadata changes to the user stories, validate the changes and submit them for approval or promote them if the option is available.
* Additionally, a lead developer should review the development work to prevent the insertion of a bad design (e.g. non-scalable triggers) in the project, which may result in errors in the long term. Then, insights can be shared with the team so that there are no surprises and developers can learn faster through feedback.

**QA Tester Tasks**:

* Make sure that everything is ready for client business stakeholders to test and then approve or reject the functionality.
* Ensure that test scripts are accurate, indicating the steps to follow in order to achieve the expected result. If user stories are well documented, QA team members can prepare tests for these user stories at the same time as they are being developed.
* Get the test script steps approved from peers and make sure they have access to the testing org with the appropriate permissions to execute the test.
* Support the business in pointing out improvements that can be done in the feature’s design.
* Support developers if issues are found, and a fix is required.

**Release Manager Tasks**:

* Move features to production and own the overall release process.
* Run continuous improvement efforts: reduce errors, increase automation and avoid wasting time.
* Help teams fix errors and resolve conflicts if any.
* Monitor and analyze the release process. What are the steps to be done by the team? Is there anything you might have missed?
* Analyze challenges. Write them down and investigate the root cause.
* Document solutions. A good documentation about best practices and how to handle specific situations will enable teams to avoid errors.
* Listen to the team, resolve any possible complaints and always aim to improve efficiency, communication and collaboration.

**Include Quality Gates**

After defining the process and the roles involved, is there anything else you should consider? The answer is yes: quality gates.

Apart from using Git integration to avoid worst-case scenarios, as you will always have a way to roll back to a stable version, you need to include additional quality gates in your development process. Depending on your business requirements, you may use:

* **Manual validations**:
  + Technical approval, such as peer review or pull request review.
  + Business testing, to confirm that business requirements are met.

﻿

* **Automated validations**:
  + Leveraging Copado features such as Static Code Analysis, Selenium Testing and Compliance Hub.
  + Salesforce features such as approval processes, validation rules, and process builders.

Quality gates are not there to annoy you, they are a safety net. Therefore, define properly which quality gates you will enforce from the beginning and include them in your documentation.

Once the whole process is defined and documented, you will be able to share it with your team so that everyone is aware of the procedure of each release stage they may be involved in.

ADAPTING COPADO TO THE PROCESS:

**Set up Copado in a Sandbox**

It is strongly recommended that you install and set up an alternative test/training implementation of Copado in a sandbox which is not part of your active development flow or pipeline in order to:

* Try different Copado features and integrations in a safe environment.
* Safely upgrade to new Copado versions to enable new features and then deploy these changes to the org where Copado is being used.
* Train new users on Copado and perform Academy module exercises.

Copado can run in a sandbox in the same way it runs in production without limitations. Just make sure you are provisioned with all the required features and licenses so that you can use Copado to its full potential.

**Adjust the User Story Layout**

Copado has included all the fields, buttons, related lists and other items that you or your team might need when working with user stories. However, you should consider modifying the main layouts with which your team will be working. Having a defined process will help you focus on what you need to adjust in the layouts. As you have already defined the process, just double-check which Copado features you and your team will be using. Then, modify the user story layout and leave only the fields, buttons, and any other items required for the process. This will simplify the layout and will allow your team to work faster and focus on other important tasks. This way, you can avoid possible misunderstandings and having fields and buttons you and your team will not use.

Are you using Static Code Analysis? If not, remove the **Run Static Code Analysis** button and the **Static Code Analysis Results** related list. Don’t worry, go ahead and remove any items you don’t need from the layout. You can easily add them back to the layout if needed.

You can also refine any other object’s layout you may use in your release process, such as projects, epics, and sprints.

**System User and Main Org Credentials**

As you are working on and deploying changes to different environments, you will need to grant Copado access to perform actions in all these environments. In order to do this, you have to create a system user and the main org credentials in each environment. You may be already familiar with the process of creating a system user, as many Salesforce implementations usually have an admin-user login, normally for integration purposes.

What about the main org credentials? Remember that an org credential is a connection between a user and a Salesforce environment. In this case, you will need to create a connection between the system user (admin-user) and the corresponding environment. This connection is what we call the main org credentials. We recommend creating main org credentials for each org and authenticating them with an admin-user login. You can name them in a way that is easy to identify such as ADMIN\_DEV or ReleaseManager\_DEV, just make sure you are consistent with your environments.

Once you have created your main org credentials, make sure you mark them as “Default”. This will avoid any risks in case the user that was used as the main Copado user leaves the project or is deactivated in all orgs.

Here are some considerations:

* If metadata is deployed with the main org credentials, the “last modified by” value in the destination will be the admin user. This is because Copado uses the org credential of the admin user to execute the deployment since it is marked as “Default”.
  + Users do not have to create their own org credentials in the destination.

﻿

* If you want the “last modified by” value of the deployed metadata to include the name of the user who executed the deployment, do not mark the main org credential of the destination as “Default”.
  + All users will have to create their own org credentials in the destination. Destination org credentials are then selected automatically when promoting a user story.

Main org credentials play an important role when setting up the Git snapshot records since they enable users to commit changes to user stories. These users will need access to an org credential that is linked to a Git snapshot record in order to be able to commit. Simply share the main org credential records of the orgs from where developers are committing changes.

**Create Public Groups**

When working with Copado, all Salesforce sharing mechanisms apply, therefore we recommend establishing sharing rules in order to reduce admin steps in user onboarding. However, sometimes there is no way to clearly identify the users with whom you should or should not share a specific org credential. As manual sharing is impossible to maintain, you should consider creating public groups so that it can be easily reverted or modified in the future.  
  
The process to follow is easy: create the required public groups, one for developers, one for lead developers, one for release managers and so on. You can name them Copado Developers, Copado Team Leads and Copado Release Managers. Then, assign users to those groups accordingly, and manually share main org credentials with those groups.  
  
**Important**: If a group is not allowed to commit from an org or deploy to that org, do not share the main org credential of that org with this group.  
  
Follow the same steps if you need to share any other Copado object with your team. As users might have multiple profiles and roles, you can group them together and provide access with just a few steps using public groups.

**Enhance Communication**

Communication in a release process is very important when multiple users are working together on their development journey. Apart from the relevant meetings in person and sharing the documentation with your team, you will also have to let them know about the changes are taking place, either by enabling Chatter in your organization or by implementing small automation. This way, they will be aware of day-to-day changes with just a few steps.  
  
When enabling Chatter in the Copado objects, you and your team will be able to track important changes in records. The Chatter feed allows you to see the details about recent changes, tag users or even leave feedback. The process to enable Chatter is the same as for any other Salesforce custom object.  
  
Additionally, you should consider implementing small automation, such as workflows or process builders that will alert users via email when changes take place (email alerts).  
  
Enhancing the communication in your DevOps processes will allow you and your team to be in sync and avoid misunderstandings.

**Set up a Backup Repository**

In addition to your working repository, it is a best practice to set up a backup repository which will not be used in your pipeline but will contain a backup of all the metadata from all your important Salesforce orgs. Follow these steps to configure your backup repository:

1. Create a new repository in your Git provider along with the relevant branches for each organization.
2. Create a new repository record in Copado.
3. Create the Git snapshot records for each branch.
4. Set the frequency of the execution to daily.

If you are not using Copado branching strategy, this will still apply and will allow you to have your metadata backed up in a Git provider. Follow the same steps mentioned before and just create the branch of the orgs you want to back up, commonly UAT and Production.

Since there are daily snapshots from Production into the backup repository, if you see new changes, this could be due to any of the following reasons:

* There was a recent release to Production that introduced the changes.
* Users are changing metadata directly in Production.
* Salesforce upgraded the org to a new version and the metadata has changed.

**Copado Developer Permission Set**

The Copado package provides the **Copado User** permission set which grants access to all Copado functionality. You can clone this permission set and customize it based on the roles: Developers, QA Testers or Scrum Master.

cdIf you want to provide your developers with more restrictive access to Copado, clone the **Copado User** permission set, name it Copado Developer and limit the access and visibility to only the required elements to streamline the navigation for your users.

There are some optional actions that you can perform to make your release processes smoother:

* Create custom Copado applications for each of the roles with only the tabs each role needs. For example:
  + Copado Development
  + Copado Release Management
  + Copado Testing

﻿

* Do not include all metadata in your working repository. Why having a snapshot of your sandboxes documents, reports and dashboards if you are managing them in production?